

Enhanced Java Interview Questions

Java Basics

1. What is the difference between JDK and JRE?
2. What is Java Virtual Machine (JVM)?
3. What are the different types of memory areas allocated by JVM?
4. What is JIT compiler?
5. How Java platform is different from other platforms?
6. Why people say that Java is 'write once and run anywhere' language?
7. How does ClassLoader work in Java?
8. Do you think 'main' used for main method is a keyword in Java?
9. Can we write main method as public void static instead of public static void?
10. In Java, if we do not specify any value for local variables, then what will be the default value of the local variables?
11. Let say, we run a java class without passing any arguments. What will be the value of String array of arguments in Main method?
12. What is the difference between byte and char data types in Java?

OOPS

13. What are the main principles of Object Oriented Programming?
14. What is the difference between Object Oriented Programming language and Object Based Programming language?
15. In Java what is the default value of an object reference defined as an instance variable in an Object?
16. Why do we need constructor in Java?
17. Why do we need default constructor in Java classes?
18. What is the value returned by Constructor in Java?
19. Can we inherit a Constructor?
20. Why constructors cannot be final, static, or abstract in Java?

Inheritance

21. What is the purpose of 'this' keyword in java?
22. Explain the concept of Inheritance?
23. Which class in Java is superclass of every other class?
24. Why Java does not support multiple inheritance?
25. In OOPS, what is meant by composition?
26. How aggregation and composition are different concepts?
27. Why there are no pointers in Java?
28. If there are no pointers in Java, then why do we get NullPointerException?
29. What is the purpose of 'super' keyword in java?
30. Is it possible to use this() and super() both in same constructor?
31. What is the meaning of object cloning in Java?

Static

32. In Java, why do we use static variable?
33. Why it is not a good practice to create static variables in Java?
34. What is the purpose of static method in Java?

- 35. Why do we mark main method as static in Java?
- 36. In what scenario do we use a static block?
- 37. Is it possible to execute a program without defining a main() method?
- 38. What happens when static modifier is not mentioned in the signature of main method?
- 39. What is the difference between static method and instance method in Java?

Method Overloading and Overriding

- 40. What is the other name of Method Overloading?
- 41. How will you implement method overloading in Java?
- 42. What kinds of argument variations are allowed in Method Overloading?
- 43. Why it is not possible to do method overloading by changing return type of method in java?
- 44. Is it allowed to overload main() method in Java?
- 45. How do we implement method overriding in Java?
- 46. Are we allowed to override a static method in Java?
- 47. Why Java does not allow overriding a static method?
- 48. Is it allowed to override an overloaded method?
- 49. What is the difference between method overloading and method overriding in Java?
- 50. Does Java allow virtual functions?
- 51. What is meant by covariant return type in Java?

Polymorphism

- 52. What is Runtime Polymorphism?
- 53. Is it possible to achieve Runtime Polymorphism by data members in Java?
- 54. Explain the difference between static and dynamic binding?

Abstraction

- 55. What is Abstraction in Object Oriented programming?
- 56. How is Abstraction different from Encapsulation?
- 57. What is an abstract class in Java?
- 58. Is it allowed to mark a method abstract method without marking the class abstract?
- 59. Is it allowed to mark a method abstract as well as final?
- 60. Can we instantiate an abstract class in Java?
- 61. What is an interface in Java?
- 62. Is it allowed to mark an interface method as static?
- 63. Why an Interface cannot be marked as final in Java?
- 64. What is a marker interface?
- 65. What can we use instead of Marker interface?
- 66. How Annotations are better than Marker Interfaces?
- 67. What is the difference between abstract class and interface in Java?
- 68. Does Java allow us to use private and protected modifiers for variables in interfaces?
- 69. How can we cast to an object reference to an interface reference?

Final

- 70. How can you change the value of a final variable in Java?
- 71. Can a class be marked final in Java?

- 72. How can we create a final method in Java?
- 73. How can we prohibit inheritance in Java?
- 74. Why Integer class is final in Java?
- 75. What is a blank final variable in Java?
- 76. How can we initialize a blank final variable?
- 77. Is it allowed to declare main method as final?

Package

- 78. What is the purpose of package in Java?
- 79. What is java.lang package?
- 80. Which is the most important class in Java?
- 81. Is it mandatory to import java.lang package every time?
- 82. Can you import same package or class twice in your class?
- 83. What is a static import in Java?
- 84. What is the difference between import static com.test.FooClass and import com.test.FooClass?

Internationalization

- 85. What is Locale in Java?
- 86. How will you use a specific Locale in Java?

Serialization

- 87. What is the serialization?
- 88. What is the purpose of serialization?
- 89. What is Deserialization?
- 90. What is Serialization and Deserialization conceptually?
- 91. Why do we mark a data member transient?
- 92. Is it allowed to mark a method as transient?
- 93. How does marking a field as transient makes it possible to serialize an object?
- 94. What is Externalizable interface in Java?
- 95. What is the difference between Serializable and Externalizable interface?

Reflection

- 96. What is Reflection in Java?
- 97. What are the uses of Reflection in Java?
- 98. How can we access private method of a class from outside the class?
- 99. How can we create an Object dynamically at Runtime in Java?

Garbage Collection

- 100. What is Garbage Collection in Java?
- 101. Why Java provides Garbage Collector?
- 102. What is the purpose of gc() in Java?
- 103. How does Garbage Collection work in Java?
- 104. When does an object become eligible for Garbage Collection in Java?
- 105. Why do we use finalize() method in Java?
- 106. What are the different types of References in Java?
- 107. How can we reference an unreferenced object again?

108.What kind of process is the Garbage collector thread?

109.What is the purpose of the Runtime class?

110. How can we invoke an external process in Java?

111. What are the uses of Runtime class?

Inner Classes

112. What is a Nested class?

113. How many types of Nested classes are in Java?

114. Why do we use Nested Classes?

115. What is the difference between a Nested class and an Inner class in Java?

116. What is a Nested interface?

117. How can we access the non-final local variable, inside a Local Inner class?

118. Can an Interface be defined in a Class?

119. Do we have to explicitly mark a Nested Interface public static?

120.Why do we use Static Nested interface in Java?

String

121. What is the meaning of Immutable in the context of String class in Java?

122.Why a String object is considered immutable in java?

123.How many objects does following code create?

124.How many ways are there in Java to create a String object?

125.How many objects does following code create?

126.What is String interning?

127.Why Java uses String literal concept?

128.What is the basic difference between a String and StringBuffer object?

129.How will you create an immutable class in Java?

130.What is the use of toString() method in java ?

131. Arrange the three classes String, StringBuffer and StringBuilder in the order of efficiency for String processing operations?

Exception Handling

132.What is Exception Handling in Java?

133.In Java, what are the differences between a Checked and Unchecked?

134.What is the base class for Error and Exception classes in Java?

135.What is a finally block in Java?

136.What is the use of finally block in Java?

137.Can we create a finally block without creating a catch block?

138.Do we have to always put a catch block after a try block?

139.In what scenarios, a finally block will not be executed?

140.Can we re-throw an Exception in Java?

141. What is the difference between throw and throws in Java?

142.What is the concept of Exception Propagation?

143.When we override a method in a Child class, can we throw an additional Exception that is not thrown by the Parent class method?

Multi-threading

- 144.How Multi-threading works in Java?**
- 145.What are the advantages of Multithreading?**
- 146.What are the disadvantages of Multithreading?**
- 147.What is a Thread in Java?**
- 148.What is a Thread's priority and how it is used in scheduling?**
- 149.What are the differences between Pre-emptive Scheduling Scheduler and Time Slicing Scheduler?**
- 150.Is it possible to call run() method instead of start() on a thread in Java?**
- 151.How will you make a user thread into daemon thread if it has already started?**
- 152.Can we start a thread two times in Java?**
- 153.In what scenarios can we interrupt a thread?**
- 154.In Java, is it possible to lock an object for exclusive use by a thread?**
- 155.How notify() method is different from notifyAll() method?**

Collections

- 156.What are the differences between the two data structures: a Vector and an ArrayList?**
- 157.What are the differences between Collection and Collections in Java?**
- 158.In which scenario, LinkedList is better than ArrayList in Java?**
- 159.What are the differences between a List and Set collection in Java?**
- 160.What are the differences between a HashSet and TreeSet collection in Java?**
- 161. In Java, how will you decide when to use a List, Set or a Map collection?**
- 162.What are the differences between a HashMap and a Hashtable in Java?**
- 163.What are the differences between a HashMap and a TreeMap?**
- 164.What are the differences between Comparable and Comparator?**
- 165.In Java, what is the purpose of Properties file?**
- 166.What is the reason for overriding equals() method?**
- 167.How does hashCode() method work in Java?**
- 168.Is it a good idea to use Generics in collections?**

Mixed Questions

- 169.What are Wrapper classes in Java?**
- 170.What is the purpose of native method in Java?**
- 171. What is System class?**
- 172.What is System, out and println in System.out.println method call?**
- 173.What is the other name of Shallow Copy in Java?**
- 174.What is the difference between Shallow Copy and Deep Copy in Java?**
- 175.What is a Singleton class?**
- 176.What is the difference between Singleton class and Static class?**

Java Collection

- 177.What is the difference between Collection and Collections Framework in Java?**
- 178.What are the main benefits of Collections Framework in Java?**
- 179.What is the root interface of Collection hierarchy in Java?**
- 180.What are the main differences between Collection and Collections?**

181. What are the Thread-safe classes in Java Collections framework?

182. How will you efficiently remove elements while iterating a Collection?

183. How will you convert a List into an array of integers like- `int[]`?

184. How will you convert an array of primitive integers `int[]` to a List collection?

185. How will you run a filter on a Collection?

186. How will you convert a List to a Set?

187. How will you remove duplicate elements from an ArrayList?

188. How can you maintain a Collection with elements in Sorted order?

189. What is the difference between `Collections.emptyList()` and creating new instance of Collection?

190. How will you copy elements from a Source List to another list?

191. What are the Java Collection classes that implement List interface?

192. What are the Java Collection classes that implement Set interface?

193. What is the difference between an Iterator and ListIterator in Java?

194. What is the difference between Iterator and Enumeration?

195. What is the difference between an ArrayList and a LinkedList data structure?

196. What is the difference between a Set and a Map in Java?

197. What is the use of a Dictionary class?

198. What is the default size of load factor in a HashMap collection in Java?

199. What is the significance of load factor in a HashMap in Java?

200. What are the major differences between a HashSet and a HashMap?

201. What are the similarities between a HashSet and a HashMap in Java?

202. What is the reason for overriding `equals()` method?

203. How can we synchronize the elements of a List, a Set or a Map?

204. What is Hash Collision? How Java handles hash-collision in HashMap?

205. What are the Hash Collision resolution techniques?

206. What is the difference between Queue and Stack data structures?

207. What is an Iterator in Java?

208. What is the difference between Iterator and Enumeration in Java?

209. What is the design pattern used in the implementation of Enumeration in Java?

210. Which methods do we need to override to use an object as key in a HashMap?

211. How will you reverse a List in Java?

212. How will you convert an array of String objects into a List?

213. What is the difference between `peek()`, `poll()` and `remove()` methods of Queue interface in java?

214. What is the difference between Array and ArrayList in Java?

215. How will you insert, delete and retrieve elements from a HashMap collection in Java?

216. What are the main differences between HashMap and ConcurrentHashMap in Java?

217. What is the increasing order of performance for following collection classes in Java?

218. Why does Map interface not extend Collection interface in Java?

219. What are the different ways to iterate elements of a list in Java?

220. What is CopyOnWriteArrayList? How it is different from ArrayList in Java?

221.How remove() method is implemented in a HashMap?

222.What is BlockingQueue in Java Collections?

223.How is TreeMap class implemented in Java?

224.What is the difference between Fail-fast and Fail-safe iterator in Java?

225.How does ConcurrentHashMap work in Java?

226.What is the importance of hashCode() and equals() methods?

227.What is the contract of hashCode() and equals() methods in Java?

228.What is an EnumSet in Java?

229.What are the main Concurrent Collection classes in Java?

230.How will you convert a Collection to SynchronizedCollection in Java?

231.How IdentityHashMap is different from a regular Map in Java?

232.What is the main use of IdentityHashMap?

233.How can we improve the performance of IdentityHashMap?

234.Is IdentityHashMap thread-safe?

235.What is a WeakHashMap in Java?

236.How can you make a Collection class read Only in Java?

237.When is UnsupportedOperationException thrown in Java?

238.Let say there is a Customer class. We add objects of Customer class to an ArrayList. How can we sort the Customer objects in ArrayList by using customer firstName attribute of Customer class?

239.What is the difference between Synchronized Collection and Concurrent Collection?

240.What is the scenario to use ConcurrentHashMap in Java?

241.How will you create an empty Map in Java?

242.What is the difference between remove() method of Collection and remove() method of Iterator?

243.Between an Array and ArrayList, which one is the preferred collection for storing objects?

244.Is it possible to replace Hashtable with ConcurrentHashMap in Java?

245.How CopyOnWriteArrayList class is different from ArrayList and Vector classes?

246.Why ListIterator has add() method but Iterator does not have?

247.Why do we sometime get ConcurrentModificationException during iteration?

248.How will you convert a Map to a List in Java?

249.How can we create a Map with reverse view and lookup in Java?

250.How will you create a shallow copy of a Map?

251.Why we cannot create a generic array in Java?

252.What is a PriorityQueue in Java?

253.What are the important points to remember while using Java Collections Framework?

254.How can we pass a Collection as an argument to a method and ensure that method will not be able to modify it?

255.Can you explain how HashMap works in Java?

256.Can you explain how HashSet is implemented in Java?

257.What is a NavigableMap in Java?

258.What is the difference between descendingKeySet() and descendingMap() methods of NavigableMap?

259.What is the advantage of NavigableMap over Map?

260.What is the difference between headMap(), tailMap() and subMap() methods of NavigableMap?

261.How will you sort objects by Natural order in a Java List?

262.How can we get a Stream from a List in Java?

263.Can we get a Map from a Stream in Java?

264.What are the popular implementations of Deque in Java?

Advanced Multi-threading

265.What is a Thread in Java?

266.What is the priority of a Thread and how it is used in scheduling?

267.What is the default priority of a thread in Java?

268.What are the three different priorities that can be set on a Thread in Java?

269.What is the purpose of join() method in Thread class?

270.What is the fundamental difference between wait() and sleep() methods?

271.Is it possible to call run() method instead of start() on a thread in Java?

272.What is a daemon thread in Java?

273.How can we make a regular thread Daemon thread in Java?

274.How will you make a user thread into daemon thread if it has already started?

275.Can we start a thread two times in Java?

276.What is a Shutdown hook in Java?

277.What is synchronization in Java?

278.What is the purpose of Synchronized block in Java?

279.What is static synchronization?

280.What is a Deadlock situation?

281.What is the meaning of concurrency?

282.What is the main difference between process and thread?

283.What is a process and thread in the context of Java?

284.What is a Scheduler?

285.What is the minimum number of Threads in a Java program?

286.What are the properties of a Java thread?

287.What are the different states of a Thread in Java?

288.How will you set the priority of a thread in Java?

289.What is the purpose of Thread Groups in Java?

290.Why we should not stop a thread by calling its stop() method?

291.How will you create a Thread in Java?

292.How can we stop a thread in the middle of execution in Java?

293.How do you access the current thread in a Java program?

294.What is Busy waiting in Multi-threading?

295.How can we prevent busy waiting in Java?

296.Can we use Thread.sleep() method for real-time processing in Java?

297.Can we wake up a thread that has been put to sleep by using Thread.sleep() method?

298.What are the two ways to check if a Thread has been interrupted?

299.How can we make sure that Parent thread waits for termination of Child thread?

300.How will you handle InterruptedException in Java?

- 301.Which intrinsic lock is acquired by a synchronized method in Java?
- 302.Can we mark a constructor as synchronized in Java?
- 303.Can we use primitive values for intrinsic locks?
- 304.Do we have re-entrant property in intrinsic locks?
- 305.What is an atomic operation?
- 306.Can we consider the statement i++ as an atomic operation in Java?
- 307.What are the Atomic operations in Java?
- 308.Can you check if following code is thread-safe?
- 309.What are the minimum requirements for a Deadlock situation in a program?
- 310.How can we prevent a Deadlock?
- 311.How can we detect a Deadlock situation?
- 312.What is a Livelock?
- 313.What is Thread starvation?
- 314.How can a synchronized block cause Thread starvation in Java?
- 315.What is a Race condition?
- 316.What is a Fair lock in multi-threading?
- 317.Which two methods of Object class can be used to implement a Producer Consumer scenario?
- 318.How JVM determines which thread should wake up on notify()?
- 319.Check if following code is thread-safe for retrieving an integer value from a Queue?
- 320.How can we check if a thread has a monitor lock on a given object?
- 321.What is the use of yield() method in Thread class?
- 322.What is an important point to consider while passing an object from one thread to another thread?
- 323.What are the rules for creating Immutable Objects?
- 324.What is the use of ThreadLocal class?
- 325.What are the scenarios suitable for using ThreadLocal class?
- 326.How will you improve the performance of an application by multithreading?
- 327.What is scalability in a Software program?
- 328.How will you calculate the maximum speed up of an application by using multiple processors?
- 329.What is Lock contention in multi-threading?
- 330.What are the techniques to reduce Lock contention?
- 331.What technique can be used in following code to reduce Lock contention?
- 332.What is Lock splitting technique?
- 333.Which technique is used in ReadWriteLock class for reducing Lock contention?
- 334.What is Lock striping?
- 335.What is a CAS operation?
- 336.Which Java classes use CAS operation?
- 337.Is it always possible to improve performance by object pooling in a multi-threading application?
- 338.How can techniques used for performance improvement in a single thread application may degrade the performance in a multi-threading application?
- 339.What is the relation between Executor and ExecutorService interface?

340.What will happen on calling submit() method of an ExecutorService instance whose queue is already full?

341.What is a ScheduledExecutorService?

342.How will you create a Thread pool in Java?

343.What is the main difference between Runnable and Callable interface?

344.What are the uses of Future interface in Java?

345.What is the difference in concurrency in HashMap and in Hashtable?

346.How will you create synchronized instance of List or Map Collection?

347.What is a Semaphore in Java?

348.What is a CountdownLatch in Java?

349.What is the difference between CountdownLatch and CyclicBarrier?

350.What are the scenarios suitable for using Fork/Join framework?

351.What is the difference between RecursiveTask and RecursiveAction class?

352.In Java 8, can we process stream operations with a Thread pool?

353.What are the scenarios to use parallel stream in Java 8?

354.How Stack and Heap work in Java multi-threading environment?

355.How can we take Thread dump in Java?

356.Which parameter can be used to control stack size of a thread in Java?

357.There are two threads T1 and T2? How will you ensure that these threads run in sequence T1, T2 in Java?

Java 8

358.What are the new features released in Java 8?

359.What are the main benefits of new features introduced in Java 8?

360.What is a Lambda expression in Java 8?

361.What are the three main parts of a Lambda expression in Java?

362.What is the data type of a Lambda expression?

363.What is the meaning of following lambda expression?

364.Why did Oracle release a new version of Java like Java 8?

365.What are the advantages of a lambda expression?

366.What is a Functional interface in Java 8?

367.What is a Single Abstract Method (SAM) interface in Java 8?

368.How can we define a Functional interface in Java 8?

369.Why do we need Functional interface in Java?

370.Is it mandatory to use @FunctionalInterface annotation to define a Functional interface in Java 8?

371.What are the differences between Collection and Stream API in Java 8?

372.What are the main uses of Stream API in Java 8?

373.What are the differences between Intermediate and Terminal Operations in Java 8 Streams?

374.What is a Spliterator in Java 8?

375.What are the differences between Iterator and Spliterator in Java 8?

376.What is Type Inference in Java 8?

377.Does Java 7 support Type Inference?

378.How does Internal Iteration work in Java 8?

379.What are the main differences between Internal and External Iterator?

- 380.What are the main advantages of Internal Iterator over External Iterator in Java 8?
- 381.What are the applications in which we should use Internal Iteration?
- 382.What is the main disadvantage of Internal Iteration over External Iteration?
- 383.Can we provide implementation of a method in a Java Interface?
- 384.What is a Default Method in an Interface?
- 385.Why do we need Default method in a Java 8 Interface?
- 386.What is the purpose of a Static method in an Interface in Java 8?
- 387.What are the core ideas behind the Date/Time API of Java 8?
- 388.What are the advantages of new Date and Time API in Java 8 over old Date API?
- 389.What are the main differences between legacy Date/Time API in Java and Date/Time API of Java 8?
- 390.How can we get duration between two dates or time in Java 8?
- 391.What is the new method family introduced in Java 8 for processing of Arrays on multi core machines?
- 392.How does Java 8 solve Diamond problem of Multiple Inheritance?
- 393.What are the differences between Predicate, Supplier and Consumer in Java 8?
- 394.Is it possible to have default method definition in an interface without marking it with default keyword?
- 395.Can we create a class that implements two Interfaces with default methods of same name and signature?
- 396.How Java 8 supports Multiple Inheritance?
- 397.In case we create a class that extends a base class and implements an interface. If both base class and interface have a default method with same name and arguments, then which definition will be picked by JVM?
- 398.If we create same method and define it in a class , in its parent class and in an interface implemented by the class, then definition will be invoked if we access it using the reference of Interface and the object of class?
- 399.Can we access a static method of an interface by using reference of the interface?
- 400.How can you get the name of Parameter in Java by using reflection?
- 401.What is Optional in Java 8?
- 402.What are the uses of Optional?
- 403.Which method in Optional provides the fallback mechanism in case of null value?
- 404.How can we get current time by using Date/Time API of Java 8?
- 405.Is it possible to define a static method in an Interface?
- 406.How can we analyze the dependencies in Java classes and packages?
- 407.What are the new JVM arguments introduced by Java 8?
- 408.What are the popular annotations introduced in Java 8?
- 409.What is a StringJoiner in Java 8?
- 410.What is the type of a Lambda expression in Java 8?
411. What is the target type of a lambda expression ?
- 412.What are the main differences between an interface with default method and an abstract class in Java 8?

Java Tricky Questions

- 413.Is there any difference between $a = a + b$ and $a += b$ expressions?
- 414.What does the expression $1.0 / 0.0$ return? Will there be any compilation error?
- 415.Can we use multiple main methods in multiple classes?
- 416.Does Java allow you to override a private or static method?
- 417.What happens when you put a key object in a HashMap that is already present?

418. How can you make sure that N threads can access N resources without deadlock?

419. How can you determine if JVM is 32-bit or 64-bit from Java Program?

420. What is the right data type to represent Money (like Dollar/Pound) in Java?

421. How can you do multiple inheritances in Java?

422. Is ++ operation thread-safe in Java?

423. How can you access a non-static variable from the static context?

424. Let say there is a method that throws NullPointerException in the superclass. Can we override it with a method that throws RuntimeException?

425. How can you mark an array volatile in Java?

426. What is a thread local variable in Java?

427. What is the difference between sleep() and wait() methods in Java?

428. Can you create an Immutable object that contains a mutable object?

429. How can you convert an Array of bytes to String?

430. What is difference between CyclicBarrier and CountdownLatch class?

431. What is the difference between StringBuffer and StringBuilder?

432. Which class contains clone method? Cloneable or Object class?

433. How will you take thread dump in Java?

434. Can you cast an int variable into a byte variable? What happens if the value of int is larger than byte?

435. In Java, can we store a double value in a long variable without explicit casting?

436. What will this return $5 * 0.1 == 0.5$? true or false?

437. Out of an int and Integer, which one takes more memory?

438. Can we use String in the switch case statement in Java?

439. Can we use multiple main methods in same class?

440. When creating an abstract class, is it a good idea to call abstract methods inside its constructor?

441. How can you do constructor chaining in Java?

442. How can we find the memory usage of JVM from Java code?

443. What is the difference between $x == y$ and $x.equals(y)$ expressions in Java?

444. How can you guarantee that the garbage collection takes place?

445. What is the relation between $x.hashCode()$ method and $x.equals(y)$ method of Object class?

446. What is a compile time constant in Java?

447. Explain the difference between fail-fast and fail-safe iterators?

448. You have a character array and a String. Which one is more secure to store sensitive data (like password, date of birth, etc.)?

449. Why do you use volatile keyword in Java?

450. What is the difference between poll() and remove() methods of Queue in Java?

451. Can you catch an exception thrown by another thread in Java?

452. How do you decide which type of Inner Class – Static or Non-Static to use in Java?

453. What are the different types of Classloaders in Java?

454. What are the situations in which you choose HashSet or TreeSet?

455. What is the use of method references in Java?

456. Do you think Java Enums are more powerful than integer constants?

457. Why do we use static initializers in Java?

458. Your client is complaining that your code is throwing `NoClassDefFoundError` or `NoSuchMethodError`, even though you are able to compile your code without error and method exists in your code. What could be the reason behind this?

459. How can you check if a `String` is a number by using regular expression?

460. What is the difference between the expressions `String s = "Temporary"` and `String s = new String("Temporary")`? Which one is better and more efficient?

461. In Java, can two equal objects have the different hash code?

462. How can we print an Array in Java?

463. Is it ok to use random numbers in the implementation of `hashCode()` method in Java?

464. Between two types of dependency injections, constructor injection and setter dependency injection, which one is better?

465. What is the difference between DOM and SAX parser in Java?

466. Between Enumeration and Iterator, which one has better performance in Java?

467. What is the difference between pass by reference and pass by value?

468. What are the different ways to sort a collection in Java?

469. Why Collection interface doesn't extend Cloneable and Serializable interfaces?

470. What is the difference between a process and a thread in Java?

471. What are the benefits of using an unordered array over an ordered array?

472. Between HashSet and TreeSet collections in Java, which one is better?

473. When does JVM call the `finalize()` method?

474. When would you use Serial Garbage collector or Throughput Garbage collector in Java?

475. In Java, if you set an object reference to null, will the Garbage Collector immediately free the memory held by that object?

476. How can you make an Object eligible for Garbage collection in Java?

477. When do you use Exception or Error in Java? What is the difference between these two?

478. What is the advantage of PreparedStatement over Statement class in Java?

479. In Java, what is the difference between throw and throws keywords?

480. What happens to the Exception object after the exception handling is done?

481. How do you find which client machine is sending request to your servlet in Java?

482. What is the difference between a Cookie and a Session object in Java?

483. Which protocol does Browser and Servlet use to communicate with each other?

484. What is HTTP Tunneling?

485. Why do we use JSP instead of Servlet in Java?

486. Is empty '.java' file name a valid source file name in Java?

487. How do you implement Servlet Chaining in Java?

488. Can you instantiate this class?

489. Why Java does not support operator overloading?

490. Why String class is Immutable or Final in Java?

491. What is the difference between sendRedirect and forward methods?

492. How do you fix your Serializable class, if it contains a member that is not serializable?

493. What is the use of run time polymorphism in Java?

- 494.What are the rules of method overloading and method overriding in Java?
- 495.What is the difference between a class and an object in Java?
- 496.Can we create an abstract class that extends another abstract class?
- 497.Why do you use Upcasting or Downcasting in Java ?
- 498.What is the reason to organize classes and interfaces in a package in Java?
- 499.What is information hiding in Java?
- 500.Why does Java provide default constructor?
- 501.What is the difference between super and this keywords in Java?
- 502.What is the advantage of using Unicode characters in Java?
- 503.Can you override an overloaded method in Java?
- 504.How can we change the heap size of a JVM?
- 505.Why should you define a default constructor in Java?
- 506.How will you make an Object Immutable in Java?
- 507.How can you prevent SQL Injection in Java Code?
- 508.Which two methods should be always implemented by HashMap key Object?
- 509.Why an Object used as Key in HashMap should be Immutable?
- 510.How can we share an object between multiple threads?
511. How can you determine if your program has a deadlock?

JSP

- 512.What are the implicit objects in JSP?
- 513.How will you extend JSP code?
- 514.How will you handle runtime exceptions in JSP?
- 515.How will you prevent multiple submits of a page that come by clicking refresh button multiple times?
- 516.How will you implement a thread safe JSP page?
- 517.How will you include a static file in a JSP page?
- 518.What are the lifecycle methods of a JSP?
- 519.What are the advantages of using JSP in web architecture?
- 520.What is the advantage of JSP over Javascript?
- 521.What is the Lifecycle of JSP?
- 522.What is a JSP expression?
- 523.What are the different types of directive tags in JSP?
- 524.What is session attribute in JSP?
- 525.What are the different scopes of a JSP object?
- 526.What is pageContext in JSP?
- 527.What is the use of jsp:useBean in JSP?
- 528.What is difference between include Directive and include Action of JSP?
- 529.How will you use other Java files of your application in JSP code?
- 530.How will you use an existing class and extend it to use in the JSP?
- 531.Why _jspService method starts with _ symbol in JSP?
- 532.Why do we use tag library in JSP?
- 533.What is the different type of tag library groups in JSTL?

- 534.How will you pass information from one JSP to another JSP?
- 535.How will you call a stored procedure from JSP?
- 536.Can we override `_jspService()` method in JSP?
- 537.What is a directive in JSP?
- 538.How will you implement Session tracking in JSP?
- 539.How do you debug code in JSP?
- 540.How will you implement error page in JSP?
- 541.How will you send XML data from a JSP?
- 542.What happens when we request for a JSP page from web browser?
- 543.How will you implement Auto Refresh of page in JSP?
- 544.What are the important status codes in HTTP?
- 545.What is the meaning of Accept attribute in HTTP header?
- 546.What is the difference between Expression and Scriptlet in JSP?
- 547.How will you delete a Cookie in JSP?
- 548.How will you use a Cookie in JSP?
- 549.What is the main difference between a Session and Cookie in JSP?
- 550.How will you prevent creation of session in JSP?
- 551.What is an output comment in JSP?
- 552.How will you prevent caching of HTML output by web browser in JSP?
- 553.How will you redirect request to another page in browser in JSP code?
- 554.What is the difference between `sendRedirect` and `forward` in a JSP?
- 555.What is the use of `config` implicit object in JSP?
- 556.What is the difference between `init-param` and `context-param`?
- 557.What is the purpose of `RequestDispatcher`?
- 558.How can be read data from a Form in a JSP?
- 559.What is a filter in JSP?
- 560.How can you upload a large file in JSP?
- 561.In which scenario, Container initializes multiple JSP/Servlet objects?

Java Design Patterns

- 562.When will you use Strategy Design Pattern in Java?
- 563.What is Observer design pattern?
- 564.What are the examples of Observer design pattern in JDK?
- 565.How Strategy design pattern is different from State design pattern in Java?
- 566.Can you explain Decorator design pattern with an example in Java?
- 567.What is a good scenario for using Composite design Pattern in Java?
- 568.Have you used Singleton design pattern in your Java project?
- 569.What are the main uses of Singleton design pattern in Java project?
- 570.Why `java.lang.Runtime` is a Singleton in Java?
- 571.What is the way to implement a thread-safe Singleton design pattern in Java?
- 572.What are the examples of Singleton design pattern in JDK?
- 573.What is Template Method design pattern in Java?

- 574.What are the examples of Template method design pattern in JDK?
- 575.Can you tell some examples of Factory Method design pattern implementation in Java?
- 576.What is the benefit we get by using static factory method to create object?
- 577.What are the examples of Builder design pattern in JDK?
- 578.What are the examples of Abstract Factory design pattern in JDK?
- 579.What are the examples of Decorator design pattern in JDK?
- 580.What are the examples of Proxy design pattern in JDK?
- 581.What are the examples of Chain of Responsibility design pattern in JDK?
- 582.What are the main uses of Command design pattern?
- 583.What are the examples of Command design pattern in JDK?
- 584.What are the examples of Interpreter design pattern in JDK?
- 585.What are the examples of Mediator design pattern in JDK?
- 586.What are the examples of Strategy design pattern in JDK?
- 587.What are the examples of Visitor design pattern in JDK?
- 588.How Decorator design pattern is different from Proxy pattern?
- 589.What are the different scenarios to use Setter and Constructor based injection in Dependency Injection (DI) design pattern?
- 590.What are the different scenarios for using Proxy design pattern?
- 591.What is the main difference between Adapter and Proxy design pattern?
- 592.When will you use Adapter design pattern in Java?
- 593.What are the examples of Adapter design pattern in JDK?
- 594.What is the difference between Factory and Abstract Factory design pattern?
- 595.What is Open/closed design principle in Software engineering?
- 596.What is SOLID design principle?
- 597.What is Builder design pattern?
- 598.What are the different categories of Design Patterns used in Object Oriented Design?
- 599.What is the design pattern suitable to access elements of a Collection?
- 600.How can we implement Producer Consumer design pattern in Java?
- 601.What design pattern is suitable to add new features to an existing object?
- 602.Which design pattern can be used when to decouple abstraction from the implementation?
- 603.Which is the design pattern used in Android applications?
- 604.How can we prevent users from creating more than one instance of singleton object by using clone() method?
- 605.What is the use of Interceptor design pattern?
- 606.What are the Architectural patterns that you have used?
- 607.What are the popular uses of Façade design pattern?
- 608.What is the difference between Builder design pattern and Factory design pattern?
- 609.What is Memento design pattern?
- 610.What is an AntiPattern?
611. What is a Data Access Object (DAO) design pattern?
- Spring Questions
- 612.What is Spring framework?

613.What are the benefits of Spring framework in software development?

614.What are the modules in Core Container of Spring framework?

615.What are the modules in Data Access/Integration layer of Spring framework?

616.What are the modules in Web layer of Spring framework?

617.What is the main use of Core Container module in Spring framework?

618.What kind of testing can be done in Spring Test Module?

619.What is the use of BeanFactory in Spring framework?

620.Which is the most popular implementation of BeanFactory in Spring?

621.What is XMLBeanFactory in Spring framework?

622.What are the uses of AOP module in Spring framework?

623.What are the benefits of JDBC abstraction layer module in Spring framework?

624.How does Spring support Object Relational Mapping (ORM) integration?

625.How does Web module work in Spring framework?

626.What are the main uses of Spring MVC module?

627.What is the purpose of Spring configuration file?

628.What is the purpose of Spring IoC container?

629.What is the main benefit of Inversion of Control (IOC) principle?

630.Does IOC containers support Eager Instantiation or Lazy loading of beans?

631.What are the benefits of ApplicationContext in Spring?

632.How will you implement ApplicationContext in Spring framework?

633.Explain the difference between ApplicationContext and BeanFactory in Spring?

634.Between ApplicationContext and BeanFactory which one is preferable to use in Spring?

635.What are the main components of a typical Spring based application?

636.Explain Dependency Injection (DI) concept in Spring framework?

637.What are the different roles in Dependency Injection (DI)?

638.Spring framework provides what kinds of Dependency Injection mechanism?

639.In Spring framework, which Dependency Injection is better? Constructor-based DI or Setter-based DI?

640.What are the advantages of Dependency Injection (DI)?

641.What are the disadvantages of Dependency Injection (DI)?

642.What is a Spring Bean?

643.What does the definition of a Spring Bean contain?

644.What are the different ways to provide configuration metadata to a Spring Container?

645.What are the different scopes of a Bean supported by Spring?

646.How will you define the scope of a bean in Spring?

647.Is it safe to assume that a Singleton bean is thread safe in Spring Framework?

648.What are the design-patterns used in Spring framework?

649.What is the lifecycle of a Bean in Spring framework?

650.What are the two main groups of methods in a Bean's lifecycle?

651.Can we override main lifecycle methods of a Bean in Spring?

652.What are Inner beans in Spring?

653.How can we inject a Java Collection in Spring framework?

654.What is Bean wiring in Spring?

655.What is Autowiring in Spring?

656.What are the different modes of Autowiring supported by Spring?

657.What are the cases in which Autowiring may not work in Spring framework?

658.Is it allowed to inject null or empty String values in Spring?

659.What is a Java-based Configuration in Spring?

660.What is the purpose of @Configuration annotation?

661.What is the difference between Full @Configuration and 'lite' @Beans mode?

662.In Spring framework, what is Annotation-based container configuration?

663.How will you switch on Annotation based wiring in Spring?

664.What is @Autowired annotation?

665.What is @Required annotation?

666.What are the two ways to enable RequiredAnnotationBeanPostProcessor in Spring?

667.What is @Qualifier annotation in Spring?

668.How Spring framework makes JDBC coding easier for developers?

669.What is the purpose of JdbcTemplate?

670.What are the benefits of using Spring DAO?

671.What are the different ways to use Hibernate in Spring?

672.What types of Object Relational Mapping (ORM) are supported by Spring?

673.How will you integrate Spring and Hibernate by using HibernateDaoSupport?

674.What are the different types of the Transaction Management supported by Spring framework?

675.What are the benefits provided by Spring Framework's Transaction Management?

676.Given a choice between declarative and programmatic Transaction Management, which method will you choose?

677.What is Aspect Oriented Programming (AOP)

678.What is an Aspect in Spring?

679.In Spring AOP, what is the main difference between a Concern and a Cross cutting concern?

680.What is a Joinpoint in Spring AOP?

681.What is an Advice in Spring AOP?

682.What are the different types of Advice in Spring AOP?

683.What is a Pointcut in Spring AOP?

684.What is an Introduction in Spring AOP?

685.What is a Target object in Spring AOP?

686.What is a Proxy in Spring AOP?

687.What are the different types of AutoProxy creators in Spring?

688.What is Weaving in Spring AOP?

689.In Spring AOP, Weaving is done at compile time or run time?

690.What is XML Schema-based Aspect implementation?

691.What is Annotation-based aspect implementation in Spring AOP?

692.How does Spring MVC framework work?

693.What is DispatcherServlet?

694.Can we have more than one DispatcherServlet in Spring MVC?

695.What is WebApplicationContext in Spring MVC?

696.What is Controller in Spring MVC framework?

697.What is @RequestMapping annotation in Spring?

698.What are the main features of Spring MVC?

699.What is the difference between a Singleton and Prototype bean in Spring?

700.How will you decide which scope- Prototype or Singleton to use for a bean in Spring?

701.What is the difference between Setter and Constructor based
Dependency Injection (DI) in Spring framework?

702.What are the drawbacks of Setter based Dependency Injection (DI) in Spring?

703.What are the differences between Dependency Injection (DI) and Factory Pattern?

704.In Spring framework, what is the difference between FileSystemResource and ClassPathResource?

705.Name some popular Spring framework annotations that you use in your project?

706.How can you upload a file in Spring MVC Application?

707.What are the different types of events provided by Spring framework?

708.What is the difference between DispatcherServlet and ContextLoaderListener in Spring?

709.How will you handle exceptions in Spring MVC Framework?

710.What are the best practices of Spring Framework?

711. What is Spring Boot?

Hibernate

712.What is Hibernate framework?

713.What is an Object Relational Mapping (ORM)?

714.What is the purpose of Configuration Interface in Hibernate?

715.What is Object Relational Impedance Mismatch?

716.What are the main problems of Object Relational Impedance Mismatch?

717.What are the key characteristics of Hibernate?

718.Can you tell us about the core interfaces of Hibernate framework?

719.How will you map the columns of a DB table to the properties of a Java class in Hibernate?

720.Does Hibernate make it mandatory for a mapping file to have .hbm.xml extension?

721.What are the steps for creating a SessionFactory in Hibernate?

722.Why do we use POJO in Hibernate?

723.What is Hibernate Query Language (HQL)?

724.How will you call a stored procedure in Hibernate?

725.What is Criteria API in Hibernate?

726.Why do we use HibernateTemplate?

727.How can you see SQL code generated by Hibernate on console?

728.What are the different types of collections supported by Hibernate?

729.What is the difference between session.save() and session.saveOrUpdate() methods in Hibernate?

730.What are the advantages of Hibernate framework over JDBC?

731.How can we get statistics of a SessionFactory in Hibernate?

732.What is the Transient state of an object in Hibernate?

733.What is the Detached state of an object in Hibernate?

734.What is the use of Dirty Checking in Hibernate?

735.What is the purpose of Callback interface in Hibernate?

736.What are the different ORM levels in Hibernate?

737.What are the different ways to configure a Hibernate application?

738.What is Query Cache in Hibernate?

739.What are the different types of Association mappings supported by Hibernate?

740.What are the different types of Unidirectional Association mappings in Hibernate?

741.What is Unit of Work design pattern?

742.In Hibernate, how can an object go in Detached state?

743.How will you order the results returned by a Criteria in Hibernate?

744.How does Example criterion work in Hibernate?

745.How does Transaction management work in Hibernate?

746.How can we mark an entity/collection as immutable in Hibernate?

747.What are the different options to retrieve an object from database in Hibernate?

748.How can we auto-generate primary key in Hibernate?

749.How will you re-attach an object in Detached state in Hibernate?

750.What is the first level of cache in Hibernate?

751.What are the different second level caches available in Hibernate?

752.Which is the default transaction factory in Hibernate?

753.What are the options to disable second level cache in Hibernate?

754.What are the different fetching strategies in Hibernate?

755.What is the difference between Immediate fetching and Lazy collection fetching?

756.What is 'Extra lazy fetching' in Hibernate?

757.How can we check is a collection is initialized or not under Lazy Initialization strategy?

758.What are the different strategies for cache mapping in Hibernate?

759.What is the difference between a Set and a Bag in Hibernate?

760.How can we monitor the performance of Hibernate in an application?

761.How can we check if an Object is in Persistent, Detached or Transient state in Hibernate?

762.What is 'the inverse side of association' in a mapping?

763.What is ORM metadata?

764.What is the difference between load() and get() method in Hibernate?

765.When should we use get() method or load() method in Hibernate?

766.What is a derived property in Hibernate?

767.How can we use Named Query in Hibernate?

768.What are the two locking strategies in Hibernate?

769.What is the use of version number in Hibernate?

770.What is the use of session.lock() method in Hibernate?

771.What inheritance mapping strategies are supported by Hibernate?

Maven

772.What is Maven?

773.What are the main features of Maven?

774.What areas of a Project can you manage by using Maven?

775.What are the main advantages of Maven?

776.Why do we say “Maven uses convention over configuration”?

777.What are the responsibilities of a Build tool like Maven?

778.What are the differences between Ant and Maven?

779.What is MOJO in Maven?

780.What is a Repository in Maven?

781.What are the different types of repositories in Maven?

782.What is a local repository in Maven?

783.What is a central repository in Maven?

784.What is a Remote repository in Maven?

785.Why we should not store jars in CVS or any other version control system instead of Maven repository?

786.Can anyone upload JARS or artifacts to Central Repository?

787.What is a POM?

788.What is Super POM?

789.What are the main required elements in POM file?

790.What are the phases in Build lifecycle in Maven?

791.What command will you use to package your Maven project?

792.What is the format of fully qualified artifact name of a Maven project?

793.What is an Archetype in Maven?

794.What is the command in Maven to generate an Archetype?

795.What are the three main build lifecycles of Maven?

796.What are the main uses of a Maven plugin?

797.How will you find the version of a plugin being used?

798.What are the different types of profile in Maven? Where will you define these profiles?

799.What are the different setting files in Maven? Where will you find these files?

800.What are the main elements we can find in settings.xml?

801.How will you check the version of Maven in your system?

802.How will you verify if Maven is installed on Windows?

803.What is a Maven artifact?

804.What are the different dependency scopes in Maven?

805.How can we exclude a dependency in Maven?

806.How Maven searches for JAR corresponding to a dependency?

807.What is a transitive dependency in Maven?

808.What are Excluded dependencies in Maven?

809.What are Optional dependencies in Maven?

810.Where will you find the class files after compiling a Maven project successfully?

811. What are the default locations for source, test and build directories in Maven?

812.What is the result of jar:jar goal in Maven?

813.How can we get the debug or error messages from the execution of Maven?

814.What is the difference between a Release version and SNAPSHOT version in Maven?

- 815.How will you run test classes in Maven?
- 816.Sometimes Maven compiles the test classes but doesn't run them? What could be the reason for it?
- 817.How can we skip the running of tests in Maven?
- 818.Can we create our own directory structure for a project in Maven?
- 819.What are the differences between Gradle and Maven?
- 820.What is the difference between Inheritance and Multi-module in Maven?
- 821.What is Build portability in Maven?

GIT

- 822.How can we see n most recent commits in GIT?
- 823.How can we know if a branch is already merged into master in GIT?
- 824.What is the purpose of git stash drop?
- 825.What is the HEAD in GIT?
- 826.What is the most popular branching strategy in GIT?
- 827.What is SubGit?
- 828.What is the use of git instaweb?
- 829.What are git hooks?
- 830.What is GIT?
- 831.What is a repository in GIT?
- 832.What are the main benefits of GIT?
- 833.What are the disadvantages of GIT?
- 834.What are the main differences between GIT and SVN?
- 835.How will you start GIT for your project?
- 836.What is git clone in GIT?
- 837.How will you create a repository in GIT?
- 838.What are the different ways to start work in GIT?
- 839.GIT is written in which language?
- 840.What does 'git pull' command in GIT do internally?
- 841.What does 'git push' command in GIT do internally?
- 842.What is git stash?
- 843.What is the meaning of 'stage' in GIT?
- 844.What is the purpose of git config command?
- 845.How can we see the configuration settings of GIT installation?
- 846.How will you write a message with commit command in GIT?
- 847.What is stored inside a commit object in GIT?
- 848.How many heads can you create in a GIT repository?
- 849.Why do we create branches in GIT?
- 850.What are the different kinds of branches that can be created in GIT?
- 851.How will you create a new branch in GIT?
- 852.How will you add a new feature to the main branch?
- 853.What is a pull request in GIT?
- 854.What is merge conflict in GIT?

855. How can we resolve a merge conflict in GIT?

856. What command will you use to delete a branch?

857. What command will you use to delete a branch that has unmerged changes?

858. What is the alternative command to merging in GIT?

859. What is Rebasing in GIT?

860. What is the 'Golden Rule of Rebasing' in GIT?

861. Why do we use Interactive Rebasing in place of Auto Rebasing?

862. What is the command for Rebasing in Git?

863. What is the main difference between git clone and git remote?

864. What is GIT version control?

865. What GUI do you use for working on GIT?

866. What is the use of git diff command in GIT?

867. What is git rerere?

868. What are the three most popular version of git diff command?

869. What is the use of git status command?

870. What is the main difference between git diff and git status?

871. What is the use of git rm command in GIT?

872. What is the command to apply a stash?

873. Why do we use git log command?

874. Why do we need git add command in GIT?

875. Why do we use git reset command?

876. What does a commit object contain?

877. How can we convert git log messages to a different format?

878. What are the programming languages in which git hooks can be written?

879. What is a commit message in GIT?

880. How GIT protects the code in a repository?

881. How GIT provides flexibility in version control?

882. How can we change a commit message in GIT?

883. Why is it advisable to create an additional commit instead of amending an existing commit?

884. What is a bare repository in GIT?

885. How do we put a local repository on GitHub server?

886. How will you delete a branch in GIT?

887. How can we set up a Git repository to run code sanity checks and UAT tests just before a commit?

888. How can we revert a commit that was pushed earlier and is public now?

889. In GIT, how will you compress last n commits into a single commit?

890. How will you switch from one branch to a new branch in GIT?

891. How can we clean unwanted files from our working directory in GIT?

892. What is the purpose of git tag command?

893. What is cherry-pick in GIT?

894. What is shortlog in GIT?

895. How can you find the names of files that were changed in a specific commit?

896.How can we attach an automated script to run on the event of a new commit by push command?

897.What is the difference between pre-receive, update and post-receive hooks in GIT?

898.Do we have to store Scripts for GIT hooks within same repository?

899.How can we determine the commit that is the source of a bug in GIT?

900.How can we see differences between two commits in GIT?

901.What are the different ways to identify a commit in GIT?

902.When we run git branch <branchname>, how does GIT know the SHA-1 of the last commit?

903.What are the different types of Tags you can create in GIT?

904.How can we rename a remote repository?

905.Some people use git checkout and some use git co for checkout. How is that possible?

906.How can we see the last commit on each of our branch in GIT?

907.Is origin a special branch in GIT?

908.How can we configure GIT to not ask for password every time?

909.What are the four major protocols used by GIT for data transfer?

910.What is GIT protocol?

911. How can we work on a project where we do not have push access?

912.What is git grep?

913.How can your reorder commits in GIT?

914.How will you split a commit into multiple commits?

915.What is filter-branch in GIT?

916.What are the three main trees maintained by GIT?

917.What are the three main steps of working GIT?

918.What are ours and theirs merge options in GIT?

919.How can we ignore merge conflicts due to Whitespace?

920.What is git blame?

921.What is a submodule in GIT?

AWS

922.What do you know about AWS Region?

923.What are the important components of IAM?

924.What are the important points about AWS IAM?

925.What are the important features of Amazon S3?

926.What is the scale of durability in Amazon S3?

927.What are the Consistency levels supported by Amazon S3?

928.What are the different tiers in Amazon S3 storage?

929.How will you upload a file greater than 100 megabytes in Amazon S3?

930.What happens to an Object when we delete it from Amazon S3?

931.What is the use of Amazon Glacier?

932.Can we disable versioning on a version-enabled bucket in Amazon S3?

933.What are the use cases of Cross Region Replication Amazon S3?

934.Can we do Cross Region replication in Amazon S3 without enabling versioning on a bucket?

935.What are the different types of actions in Object Lifecycle Management in Amazon S3?

936.How do we get higher performance in our application by using Amazon CloudFront?

937.What is the mechanism behind Regional Edge Cache in Amazon CloudFront?

938.What are the benefits of Streaming content?

939.What is Lambda@Edge in AWS?

940.What are the different types of events triggered by Amazon CloudFront?

941.What is Geo Targeting in Amazon CloudFront?

942.What are the main features of Amazon CloudFront?

943.What are the security mechanisms available in Amazon S3?

Cloud Computing

944.What are the benefits of Cloud Computing?

945.What is On-demand computing in Cloud Computing?

946.What are the different layers of Cloud computing?

947.What resources are provided by Infrastructure as a Service (IAAS) provider?

948.What is the benefit of Platform as a Service?

949.What are the main advantages of PaaS?

950.What is the main disadvantage of PaaS?

951.What are the different deployment models in Cloud computing?

952.What is the difference between Scalability and Elasticity?

953.What is Software as a Service?

954.What are the different types of Datacenters in Cloud computing?

955.Explain the various modes of Software as a Service (SaaS) cloud environment?

956.What are the important things to care about in Security in a cloud environment?

957.Why do we use API in cloud computing environment?

958.What are the different areas of Security Management in cloud?

959.What are the main cost factors of cloud based data center?

960.How can we measure the cloud-based services?

961.How a traditional datacenter is different from a cloud environment?

962.How will you optimize availability of your application in a Cloud environment?

963.What are the requirements for implementing IaaS strategy in Cloud?

DOCKER

964.What is Docker?

965.What is the difference between Docker image and Docker container?

966.How will you remove an image from Docker?

967.How is a Docker container different from a hypervisor?

968.Can we write compose file in json file instead of yaml?

969.Can we run multiple apps on one server with Docker?

970.What are the common use cases of Docker?

971.What are the main features of Docker-compose?

972.What is the most popular use of Docker?

973.What is the role of open source development in the popularity of Docker?

UNIX Shell

- 974.How will you remove all files in current directory? Including the files that are two levels down in a sub-directory.
- 975.What is the difference between the `-v` and `-x` options in Bash shell scripts?
- 976.What is a Filter in Unix command?
- 977.What is Kernel in Unix operating system?
- 978.What is a Shell in Unix OS?
- 979.What are the different shells in Unix that you know about?
- 980.What is the first character of the output in `ls -l` command ?
- 981.What is the difference between Multi-tasking and Multi-user environment?
- 982.What is Command Substitution in Unix?
- 983.What is an Inode in Unix?
- 984.What is the difference between absolute path and relative path in Unix file system?
- 985.What are the main responsibilities of a Unix Shell?
- 986.What is a Shell variable?
- Microservices**
- 987.What is Microservice?
- 988.What are the benefits of Microservices architecture?
- 989.What is the role of architect in Microservices architecture?
- 990.What is the advantage of Microservices architecture over Service Oriented Architecture (SOA)?
- 991.Is it a good idea to provide a Tailored Service Template for Microservices development in an organization?
- 992.What are the disadvantages of using Shared libraries approach to decompose a monolith application?
- 993.What are the characteristics of a Good Microservice?
- 994.What is Bounded Context?
- 995.What are the points to remember during integration of Microservices?
- 996.Is it a good idea for Microservices to share a common database?
- 997.What is the preferred type of communication between Microservices? Synchronous or Asynchronous?
- 998.What is the difference between Orchestration and Choreography in microservices architecture?
- 999.What are the issues in using REST over HTTP for Microservices?
1000. Can we create Microservices as State Machines?

1. What is the difference between JDK and JRE?

JDK stands for Java Development Kit. It contains the tools and libraries for development of Java programs. It also contains compilers and debuggers needed to compile Java program, JRE stands for Java Runtime Environment. This is included in JDK. JRE provides libraries and JVM that are required to run a Java program.

2. What is Java Virtual Machine (JVM)?

Java Virtual Machine (JVM) is an abstract machine that executes Java Bytecode. There are different JVMs for different hardware and software platforms, making JVM platform dependent. JVM is responsible for loading, verifying, and executing the Bytecode on a platform.

3. What are the different types of memory areas allocated by JVM?

In Java, JVM allocates memory to different processes, methods, and objects. Some of the memory areas allocated by JVM are:

- **ClassLoader:** It is a component of JVM used to load class files.
- **Class (Method) Area:** It stores per-class structures such as the runtime constant pool, field and method data, and the code for methods.
- **Heap:** Heap is created at runtime, and it contains the runtime data area in which objects are allocated.
- **Stack:** Stack stores local variables and partial results at runtime. It also helps in method invocation and return value. Each thread creates a private JVM stack at the time of thread creation.
- **Program Counter Register:** This memory area contains the address of the Java Virtual Machine instruction that is currently being executed.
- **Native Method Stack:** This area is reserved for all the native methods used in the application.

4. What is JIT compiler?

Just In Time compiler, also known as JIT compiler, is used for performance improvement in Java. It is enabled by default. It is compilation done at execution time rather than earlier. Java has popularized the use of JIT compiler by including it in JVM.

5. How is Java platform different from other platforms?

Java is a platform-independent language. Java compiler converts Java code into byte code that can be interpreted by JVM. There are JVMs written for almost all the popular platforms in the world. Java byte code can run on any supported platform in the same way, whereas other languages require libraries compiled for a specific platform to run.

6. Why do people say that Java is a 'write once and run anywhere' language?

You can write Java code on Windows and compile it on the Windows platform. The class and jar files that you get from Windows can run as they are on a Unix environment. So it is a truly platform-independent language. Behind all this portability is Java byte code. Byte code generated by the Java compiler can be interpreted by any JVM, making it much easier to write programs in Java and expect those to run on any platform. Java compiler `javac` compiles java code, and JVM `java` runs that code.

7. How does ClassLoader work in Java?

In Java, ClassLoader is a class that is used to load files in JVM. ClassLoader loads files from their physical file locations, such as the filesystem or network locations. There are three main types of ClassLoaders in Java: Bootstrap ClassLoader, which is the first ClassLoader and loads classes from `rt.jar` file; Extension ClassLoader, which loads class files from the `jre/lib/ext` location; and Application ClassLoader, which depends on `CLASSPATH` to find the location of class files. If you specify your jars in `CLASSPATH`, then this ClassLoader will load them.

8. Do you think 'main' used for the main method is a keyword in Java?

No, `main` is just a name of a method. There can be multiple methods with the same name `main` in a class file. It is not a keyword in Java.

9. Can we write main method as public void static instead of public static void?

No, you cannot write it like this. Any method has to first specify the modifiers and then the return value. The order of modifiers can change. We can write static public void main() instead of public static void main().

10. In Java, if we do not specify any value for local variables, then what will be the default value of the local variables?

Java does not initialize local variables with any default value. So these variables will be just null by default.

11. Let's say we run a Java class without passing any arguments. What will be the value of the String array of arguments in the main method?

By default, the value of the String array of arguments is empty in Java. It is not null.

12. What is the difference between byte and char data types in Java?

Both byte and char are numeric data types in Java. They are used to represent numbers in a specific range. The major difference between them is that a byte can store raw binary data, whereas a char stores characters or text data. Usage of char is, for example: char ch = 'x'; Byte values range from -128 to 127. A byte is made of 8 bits, but a char is made of 16 bits, so it is equivalent to 2 bytes.

13. What are the main principles of Object Oriented Programming?

Main principles of Object Oriented Programming (OOPS) are: 1. Abstraction 2. Encapsulation 3. Inheritance 4. Polymorphism

14. What is the difference between Object Oriented Programming language and Object Based Programming language?

Object Oriented Programming languages like Java and C++ follow concepts of OOPS such as Encapsulation, Abstraction, Polymorphism, and Inheritance. Object Based Programming languages follow some features of OOPS but do not provide support for Polymorphism and Inheritance. Examples include JavaScript and VBScript. Object Based Programming languages support objects, allowing you to build objects from constructors and also support Encapsulation. These languages are also known as Prototype-oriented languages.

15. In Java what is the default value of an object reference defined as an instance variable in an Object?

All the instance variable object references in Java are null.

16. Why do we need constructor in Java?

Java is an object-oriented language in which we create and use objects. A constructor is a piece of code similar to a method, used to create an object and set its initial state. A constructor is a special function that has the same name as the class. Without a constructor, there is no other way to create an object. By default, Java provides a default constructor for every object. If we overload a constructor, then we must implement the default constructor ourselves.

17. Why do we need default constructor in Java classes?

The default constructor is a no-argument constructor automatically generated by Java if no other constructor is defined. The Java specification states that it provides a default constructor if there is no overloaded constructor in a class, but does not specify what happens if an overloaded constructor is present. We need at least one constructor to create an object, which is why Java provides a default constructor. When we have an overloaded constructor, Java assumes we want custom behavior, so it does not provide a default constructor, which can result in an error if one is needed.

18. What is the value returned by Constructor in Java?

When a constructor is called in Java, it returns the object created by it. This is how new objects are instantiated.

19. Can we inherit a Constructor?

No, Java does not support inheritance of constructors.

20. Why constructors cannot be final, static, or abstract in Java?

If a method is marked as final, it cannot be overridden, but constructors cannot be overridden according to the Java Language Specification, so marking them final is unnecessary. If a method is abstract, it has no body and should be implemented in a child class; however, constructors are called implicitly when the new keyword is used and must have a

body. If a method is static, it belongs to the class rather than a specific object, but constructors are always used to initialize an object, making marking them static inappropriate.

21. What is the purpose of 'this' keyword in Java?

In Java, 'this' keyword refers to the current instance of the object. It is useful for differentiating between instance variables and local variables. It can be used to call constructors or to refer to the instance. In case of method overriding, 'this' is used for calling the method of the current class.

22. Explain the concept of Inheritance?

Inheritance is an important concept in Object Oriented Programming. Some objects share certain characteristics and behavior. By using inheritance, we can put the common behavior and characteristics in a base class, which is also known as a super class. Then all the objects with common behavior inherit from this base class. It is also represented by IS-A relationship. Inheritance promotes code reuse, method overriding, and polymorphism.

23. Which class in Java is superclass of every other class?

Java is an object oriented programming language. In Java, Object class is the superclass of every other class.

24. Why does Java not support multiple inheritance?

Multiple inheritance means that a class can inherit behavior from two or more parent classes. The issue with multiple inheritance is that both the parent classes may have different implementations for the same method, so they have different ways of doing the same thing. Now, which implementation should the child class choose? This leads to ambiguity in multiple inheritance. This is the main reason for Java not supporting multiple inheritance in implementation. For example, if you have a class TV and another class AtomBomb, both have a method switchOn(), but only TV has switchOff() method. If your class inherits from both these classes, then you have an issue that you can switchOn() both parents, but switchOff will only switch off the TV. However, you can implement multiple interfaces in Java.

25. In OOPS, what is meant by composition?

Composition is also known as a "has-a" relationship. In composition, "has-a" relation relates two classes. For example, class Car has a steering wheel. If a class holds the instance of another class, then it is called composition.

26. How are aggregation and composition different concepts?

In OOPS, aggregation and composition are types of association relations. Composition is a strong relationship. If the composite object is destroyed, then all its parts are destroyed. For example, a Car has a Steering Wheel. If the Car object is destroyed, then there is no meaning of Steering Wheel. In aggregation, the relationship is weaker than composition. For example, a Library has students. If a Library is destroyed, students still exist, so Library and Student are related by aggregation. A Library has Books. If the Library is destroyed, the Books are also destroyed. Books of a Library cannot exist without the Library, so Book and Library are related by composition.

27. Why are there no pointers in Java?

In Java, there are references instead of pointers. These references point to objects in memory, but there is no direct access to these memory locations. JVM is free to move the objects within VM memory. The absence of pointers helps Java in managing memory and garbage collection effectively. It also provides developers with the convenience of not worrying about memory allocation and de-allocation.

28. If there are no pointers in Java, then why do we get NullPointerException?

In Java, the pointer equivalent is object reference. When we use a dot (.) it points to object reference, so JVM uses pointers but programmers only see object references. In case an object reference points to null object, and we try to access a method or member variable on it, then we get NullPointerException.

29. What is the purpose of 'super' keyword in Java?

'Super' keyword is used in the methods or constructors of a child class. It refers to the immediate parent class of an object. By using 'super' we can call a method of the parent class from the method of a child class. We can also call the constructor of a parent class from the constructor of a child class by using 'super' keyword.

30. Is it possible to use this() and super() both in the same constructor?

No, Java does not allow us to use both super() and this() in the same constructor. As per Java specification, super() or this() must be the first statement in a constructor.

31. What is the meaning of object cloning in Java?

Object.clone() method is used for creating an exact copy of the object in Java. It acts like a copy constructor. It creates and returns a copy of the object, with the same class and with all the fields having same values as of the original object. One disadvantage of cloning is that the return type is an Object. It has to be explicitly cast to actual type.

32. In Java, why do we use static variable?

Whenever we want to have a common property for all objects of a class, we use a class level variable i.e. a static variable. This variable is loaded in memory only once at the time of class loading. So it saves memory, since it is not defined per object in Java.

33. Why is it not a good practice to create static variables in Java?

Static variables are common to all the objects of a class. If a new object is created, there is no need to test the value of static variable. Any code that uses static variable can be in any state. It can be within a new object or at a class level. So the scope of static variable is open ended in a Java class. If we want tighter control on scope, then variables should be created at the object creation level. Also defining static variables is not a good practice because they go against the principles of Object Oriented Programming.

34. What is the purpose of static method in Java?

Java provides the feature of static method to create behavior at the class level. The static method is common to all the objects of a class. We do not need to create any object of a class to call a static method. So it provides convenience of not creating an object for calling it. Also a static method can access and modify static data members. This also helps in keeping the behavior as well as state at the class level.

35. Why do we mark main method as static in Java?

The main method in Java is marked as static, so that JVM can call it to start the program. If main method is not static, then which constructor will be called by Java process? As such it is a known as convention to mark main method static in Java. But if we remove the static, then there will be ambiguity. Java process may not know which method of a class to call to start the program. So this convention helps in Java process to identify the starting code for a program in class that is passed as an argument to java process.

36. In what scenario do we use a static block?

At times, there is a class that has static member variables. These variables need some complicated initialization. At this time static block helps as a tool to initialize complex static member variable initialization. The static block is executed even before the execution of main. Sometimes, we can also replace static block with a static method of class.

37. Is it possible to execute a program without defining a main() method?

No, with Java 7 onwards, you need a main() method to execute a program. In earlier versions of Java, there was a workaround available to use static blocks for execution. But now this gap has been closed.

38. What happens when static modifier is not mentioned in the signature of main method?

As per Java specification, main method has to be marked as static. It needs only one argument that is an array of String. A program can compile with a non-static method. But on execution it will give NoSuchMethodError.

39. What is the difference between static method and instance method in Java?

Often, there is a need to define a behavior for a class that is not dependent on member variables of an object. Such behavior is captured in a static method. If there is a behavior dependent upon the member variables of an object, then we do not mark it static, it remains as instance method. To call as static method, we do not need to create an object. We just

call it with class name. But to call an instance method, we need to create/get an object first. Instance member variables cannot be accessed by a static method. But an instance method can call both instance variables and static variables.

40. What is the other name of Method Overloading?

Method Overloading is also known as Static Polymorphism.

41. How will you implement method overloading in Java?

In Java, a class can have multiple methods with same name but different arguments. It is called Method Overloading. To implement method overloading we have to create two methods with same name in a class and do one/more of the following: Different number of parameters, Different data type of parameters, Different sequence of data type of parameters.

42. What kinds of argument variations are allowed in Method Overloading?

Method Overloading allows two methods with same name to differ in: Number of parameters, Data type of parameters, Sequence of data type of parameters.

43. Why is it not possible to do method overloading by changing return type of method in Java?

If we change the return type of overloaded methods then it will lead to ambiguous behavior. How will clients know which method will return what type. Due to this different return type are not allowed in overloaded methods.

44. Is it allowed to overload main() method in Java?

Yes, Java allows users to create many methods with same name 'main'. But only public static void main(String[] args) method is used for execution.

45. How do we implement method overriding in Java?

To override a method, we just provide a new implementation of a method with same name in subclass. So there will be at least two implementations of the method with same name. One implementation is in parent class. And another implementation is in child class.

46. Are we allowed to override a static method in Java?

No. Java does not allow overriding a static method. If you create a static method with same name in subclass, then it is a new method, not an overridden method.

47. Why does Java not allow overriding a static method?

To override a method, you need an instance of a class. Static method is not associated with any instance of the class. So the concept of overriding does not apply here. Therefore, Java does not allow overriding a static method.

48. Is it allowed to override an overloaded method?

Yes. You can override an overloaded method in Java.

49. What is the difference between method overloading and method overriding in Java?

Differences between method overloading and overriding are: Method overloading is static polymorphism. Method overriding is runtime polymorphism. Method overloading occurs within the same class. Method overriding happens in two classes with hierarchy relationship. Parameters must be different in method overloading. Parameters must be same in method overriding. Method overloading is a compile time concept. Method overriding is a runtime concept.

50. Does Java allow virtual functions?

Yes. All instance methods in Java are virtual functions by default. Only class methods and private instance methods are not virtual methods in Java.

51. What is meant by covariant return type in Java?

A covariant return type of a method is one that can be replaced by a "narrower" type when the method is overridden in a subclass. Let's say class B is child of class A. There is a get() method in class A as well as class B. get() method of class A can return an instance of A, and get() method of class B return an instance of B. Here class B overrides get() method, but the return type is different. Before Java 5, any method that overrides the method of parent class would have same return type.

From Java 5 onwards, a child class can override a method of parent class and the child class method can return an object that is child of object return by parent class method.

52. What is Runtime Polymorphism?

Runtime Polymorphism or Dynamic Polymorphism is the polymorphism that exists at runtime. In case of method overriding it is not known which method will be called at runtime. Based on the type of object, JVM decides the exact method that should be called. So at compile time it is not known which method will be called at run time.

53. Is it possible to achieve Runtime Polymorphism by data members in Java?

No. We need to create Runtime Polymorphism by implementing methods at two levels of inheritance in Java.

54. Explain the difference between static and dynamic binding?

In Static binding references are resolved at compile time. In Dynamic binding references are resolved at Run time.

```
1 // Example of compile-time binding
2 Person p = new Person();
3 p.walk(); // Java compiler resolves this binding at compile time
4
5 // Example of dynamic binding
6 public void walk(Object o) {
7     ((Person) o).walk(); // this is dynamic binding
8 }
```

55. What is Abstraction in Object Oriented programming?

Abstraction is the process of hiding certain implementation details of an object and showing only essential features of the object to outside world. It is different from Abstract class in Java. Abstraction process identifies commonalities and hides the complexity of implementation. It helps us in focusing on the interface that we share with the outside world.

56. How is Abstraction different from Encapsulation?

Abstraction happens at class level design. It results in hiding the implementation details. Encapsulation is also known as “Information Hiding”. An example of encapsulation is marking the member variables private and providing getter and setter for these member variables.

57. What is an abstract class in Java?

An abstract class in Java has one or more abstract methods. An abstract method is just declared in the abstract class, but it is not implemented. An abstract class has to be extended in Java and its abstract methods have to be implemented by a child class. Also Java does not allow new instance of Abstract class.

58. Is it allowed to mark a method abstract method without marking the class abstract?

No. Java specification says that if there is at least one abstract method in a class, the class has to be marked abstract.

59. Is it allowed to mark a method abstract as well as final?

No. It will be contradictory statement to mark a method abstract as well as final. An abstract method has to be overridden by a child class. And a final method cannot be overridden. Therefore a method can be either abstract or final in Java.

60. Can we instantiate an abstract class in Java?

No. We cannot create an instance of an abstract class in Java.

61. What is an interface in Java?

An Interface in Java is an abstract type blueprint of a class. It contains the methods that a class must implement. It is like a protocol. It has method signatures and constant declarations.

62. Is it allowed to mark an interface method as static?

Yes, from Java 8 onwards, we can define static and default methods in an interface. Prior to Java 8, it was not allowed.

63. Why an Interface cannot be marked as final in Java?

A final method cannot be overridden. But an interface method has to be implemented by another class. So the interface method cannot be marked as final.

64. What is a marker interface?

There are interfaces that do not have any data member or methods. These interfaces are called Marker interface. E.g. Serializable, Cloneable, Remote etc.

65. What can we use instead of Marker interface?

We can use annotations instead of Marker interface.

66. How Annotations are better than Marker Interfaces?

Annotations serve the purpose of conveying metadata about the class to its consumers without creating a separate type for it. Annotations are more powerful than a Marker interface. They allow programmers to pass more sophisticated information on to classes that "consume" it.

67. What is the difference between abstract class and interface in Java?

Differences between Abstract class and Interface are as follows:

- An abstract class can have implemented methods with body (non-abstract methods). Interface has only abstract methods. From Java 8 onwards, interface can have static/default methods in implemented form.
- An abstract class can have instance member variables. An interface cannot have instance variables. It can only have constants.
- An abstract class can have a constructor. Interface cannot have constructor. It has to be implemented by another class.
- A class can extend only one abstract class. A class can implement more than one interface.

68. Does Java allow us to use private and protected modifiers for variables in interfaces?

No. All the variables in an interface are implicitly public.

69. How can we cast to an object reference to an interface reference?

An Object that implements an Interface can be cast to the same Interface. Since An Object implementing an Interface already provides implementation for the methods of that Interface, it is allowed to do so as per the rules of Inheritance.

70. How can you change the value of a final variable in Java?

Java does not allow changing the value of a final variable. Once the value is set, it cannot be changed.

71. Can a class be marked final in Java?

Yes, a class can be marked final in Java. Once a class is marked final, it cannot be extended.

72. How can we create a final method in Java?

To mark a method, add modifier final to that method. A final method can not be overridden by a child class.

73. How can we prohibit inheritance in Java?

If you mark a class final, it cannot be extended. This will prohibit the inheritance of that class in Java.

74. Why Integer class is final in Java?

Integer class is a wrapper for int. If it is not marked final, then any other class can extend it and modify the behavior of Integer operations. To avoid this, Integer wrapper class is marked as final.

75. What is a blank final variable in Java?

When we declare a final variable without giving any initial value, then it is called blank final variable.

76. How can we initialize a blank final variable?

A blank final instance variable can be initialized in a constructor. A blank final static variable can be initialized in the static block of the class.

77. Is it allowed to declare main method as final?

Yes, we can mark the main method as final.

78. What is the purpose of package in Java?

A package is used to encapsulate a group of classes, interfaces, and sub-packages. Often, it is a hierarchical structure of storing information. It is easier to organize the related classes and sub-packages in this manner. A package also provides access protection for classes and interfaces. A package also helps in removing naming collision.

79. What is java.lang package?

In Java, java.lang package contains the classes that are fundamental to the design of Java programming language. The most important class in this package is Object class. It also contains wrapper classes like Integer, Boolean, Character, etc. It provides Math class for mathematical operations.

80. Which is the most important class in Java?

It is an open-ended question with many answers. In my view, Object class is the most important class of Java programming language. It is the root of all the classes in Java. It provides some very important and fundamental methods.

81. Is it mandatory to import java.lang package every time?

No. By default, JVM loads it internally.

82. Can you import same package or class twice in your class?

If we import same package multiple times in a class, compiler includes it only once. So neither JVM nor Compiler gives any error/warning on including a package multiple times. If you have two classes with same name, then you may get name collision on importing the class erroneously. JVM internally loads the class only one time.

83. What is a static import in Java?

Static import is similar to normal import declaration. Normal import allows us to import classes from packages without using package qualifier. Static import allows us to import static members from a class without using class qualifier.

84. What is the difference between import static com.test.Fooclass and import com.test.Fooclass?

First import is a static import and the second import is normal import of a class. First import allows us to import static members of class.

85. What is Locale in Java?

A Locale object represents a specific geographical, political, or cultural region. It is used to locale-sensitive operations in Java. It helps in following the local conventions of a country, native, or region. These conventions can be for formatting the dates, money, numbers, etc.

86. How will you use a specific Locale in Java?

To use a specific Locale, we need to load that Locale. We can use ResourceBundle.getBundle("Locale.UK") method to load a Locale.

87. What is the serialization?

Serialization is a process converting an object into a byte array. This byte array represents the class, version, and internal state of the object. JVM can use this byte array to transmit/read the object over a network.

88. What is the purpose of serialization?

Some of the uses of serialization are:

- Communication: It is used for transmitting an object over network between two machines.
- Persistence: We can store the object's state in a database and retrieve it from database later on.
- Caching: Serialization can be used for caching to improve performance. We may need 10 minutes to build an object, but it may take just 10 seconds to de-serialize the object.
- Cross JVM Synchronization: It can be used in same way across multiple JVM that follow different architecture.

89. What is Deserialization?

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse process of serialization.

90. What is Serialization and Deserialization conceptually?

Serialization is to convert Object data into a stream of bytes. Deserialization is to convert a stream of bytes back into a copy of the original object.

91. Why do we mark a data member transient?

Member variables of an object are marked transient to indicate that they should not be serialized. During serialization process the transient variables are not considered part of the persistent state of an object.

92. Is it allowed to mark a method as transient?

No, Java does not allow marking a method as transient. The transient keyword is valid only for member variables.

93. How does marking a field as transient makes it possible to serialize an object?

Let's say we have a class ABC that implements Serializable interface, but it contains a member variable object of class XYZ that does not implement Serializable interface. Due to this it is not possible to Serialize the class ABC. To solve this issue, we can mark the member variable XYZ as transient in class ABC. This will allow us to serialize the class ABC.

94. What is Externalizable interface in Java?

Externalizable interface extends Serializable interface in Java. It is used for giving the Class control over saving and restoring the contents of its instances. A class implements methods writeExternal() and readExternal() to store and restore the object.

95. What is the difference between Serializable and Externalizable interface?

Serializable is a marker interface but Externalizable is not a marker interface. When we implement Serializable interface, the class is serialized automatically by default. We can override writeObject() and readObject() methods to control more complex object Serialization process. In case of Externalizable, we use readExternal() and writeExternal() methods to give control to class for class's serialization process. Serializable interface is based on recursive algorithm. Serializable gives you two options. One option is to provide custom way of serialization, the other default way. In Externalizable, you have to always implement readExternal() and writeExternal() methods. A public no-arg constructor is needed while using Externalizable interface. In Serialization, we need to define serialVersionUID. If it is not explicitly defined it will be generated automatically based on all the fields, methods of the class.

96. What is Reflection in Java?

Reflection is Java language's ability to inspect and dynamically call classes, methods, attributes etc. at Runtime. It helps in examining or modifying the Runtime behavior of a class at Runtime.

97. What are the uses of Reflection in Java?

Reflection is often used in Testing, Debugging and in Integrated Development Environment (IDE). Reflection allows you to write programs that do not have to "know" everything at compile time. It makes programs more dynamic, since they can be tied together at runtime. Many modern frameworks like Spring etc. use Reflection. Some modern languages like Python etc. also use Reflection. JAVA API for XML Parsing (JAXP) also uses Reflection.

98. How can we access private method of a class from outside the class?

We can use Reflection to access a private method of a class from outside the class. In Java, we use getDeclaredMethod() to get an instance of a private method. Then we mark this method accessible and finally invoke it. In the following sample code, we are accessing the private method message() of class Foo by Reflection.

99. How can we create an Object dynamically at Runtime in Java?

We can use Reflection to create an Object dynamically at Runtime in Java. We can use Class.newInstance() or Constructor.newInstance() methods for creating such Objects.

Garbage Collection

100. What is Garbage Collection in Java?

Java has an internal mechanism called Garbage Collection to reclaim the memory of unused objects at runtime. Garbage Collection is also known as automatic memory management.

101. Why does Java provide a Garbage Collector?

In Java, there are no pointers. Memory management and allocation are done by the JVM. Since memory allocation is automated, after some time the JVM may go low on memory. At that time, the JVM has to free memory from unused objects. To help with the process of reclaiming memory, Java provides an automated process called the Garbage Collector.

102. What is the purpose of gc() in Java?

Java provides two methods, `System.gc()` and `Runtime.gc()`, to request the JVM to run the garbage collection. By using these methods, programmers can explicitly send a request for Garbage Collection. However, the JVM can reject this request and wait for some time before running the GC.

103. How does Garbage Collection work in Java?

Java has an automated process called the Garbage Collector for memory management. It is a daemon in the JVM that monitors memory usage and performs memory cleanup. Once the JVM is low on memory, the GC process finds the unused objects that are not referenced by other objects. These unused objects are cleaned up by the Garbage Collector daemon in the JVM.

104. When does an object become eligible for Garbage Collection in Java?

An object can be garbage collected by the JVM if it is not reachable. There are two cases for deciding the eligibility of objects for Garbage Collection:

- An object/instance that cannot be reached by a live thread.
 - A set of circularly referenced instances that cannot be reached by any other instance outside that set.
-

105. Why do we use the finalize() method in Java?

Java provides the `finalize()` method to perform any cleanup before Garbage Collection. This method is in the `Object` class, and it is invoked by the JVM internally. Developers are free to implement this method for any custom cleanup in case of Garbage Collection. If an object is not garbage collected, then this method may not be called. This method is never invoked more than once by the JVM.

106. What are the different types of references in Java?

In Java, there are four types of references:

- Strong Reference
 - Soft Reference
 - Weak Reference
 - Phantom Reference
-

107. How can we reference an unreferenced object again?

We can provide an implementation in the `finalize()` method to reference an unreferenced object. For an unreferenced object, the `finalize()` method is called at the time of Garbage Collection. At this time, the object can pass its reference 'this' to the `finalize()` method and revive itself.

108. What kind of process is the Garbage Collector thread?

Garbage Collection is a daemon process in the JVM. It is an internal process that keeps checking memory usage and cleans up the memory.

109. What is the purpose of the Runtime class?

The purpose of the Runtime class is to provide access to the Java Runtime system. This class provides certain important methods like:

- `Runtime.freeMemory()` – This method returns the value of free memory in the JVM.
 - `Runtime.maxMemory()` – This method returns the value of maximum memory that the JVM can use.
 - `Runtime.gc()` – This method can invoke garbage collection.
-

110. How can we invoke an external process in Java?

Java provides the method `Runtime.getRuntime().exec()` to invoke an external process from the JVM.

111. What are the uses of the Runtime class?

The Runtime class in Java provides the following benefits:

- It allows reading data via keyboard.
 - It can use system properties and environment variables.
 - It helps in running non-Java programs from within a Java application.
-

112. What is a Nested class?

In Java, a nested class is a class declared inside another class. We can have more than one class declared inside a file.

113. How many types of Nested classes are in Java?

Java provides four types of nested classes:

- Member inner class
 - Local inner class
 - Anonymous inner class
 - Static nested class
-

114. Why do we use Nested Classes?

There are following reasons for using nested classes:

- **Logical Grouping:** We can logically group classes in one place. If one class is useful to only one other class, then we put the smaller class within the larger class and keep them in one file. This kind of nesting "helper classes" in a top-level class makes the package more streamlined.
 - **Encapsulation:** Nested classes increase encapsulation. Let's say there are two top-level classes, Foo and Bar. Bar needs access to private members of Foo. We can hide class Bar within class Foo. In this way, private members of Foo can be accessed by class Bar. So class Foo remains encapsulated. Also, class Bar remains hidden from the outside world.
 - **Code Clarity:** Nested classes make the code more readable and well-organized. Only top-level classes are exposed. The helper classes are kept hidden and closer to the code where they are used by a top-level class.
-

115. What is the difference between a Nested class and an Inner class in Java?

An Inner class in Java is a non-static class. It is a type of Nested class that is defined in another class but not qualified with a static modifier. A Nested class can be either a static nested class or a non-static inner class. An Inner class has access to other members of the enclosing class, even if they are declared private. A static nested class cannot access the other members of the enclosing class.

116. What is a Nested interface?

A Nested interface is declared inside another interface or a top-level class. By default, it is static. A Nested interface is also known as a Static interface.

117. How can we access the non-final local variable, inside a Local Inner class?

Java allows a Local Inner class to access only constant local members. So we have to make the non-final local variable a final constant to access it inside a Local Inner class.

118. Can an Interface be defined in a Class?

Yes, we can define a static nested interface within a class. Only the enclosing class can access it.

119. Do we have to explicitly mark a Nested Interface public static?

A Nested Interface is implicitly public static. So the modifiers public and static are redundant in the declaration.

120. Why do we use Static Nested interface in Java?

Only the enclosing class can access a static nested interface. Consider the following code in which interface Xyz is enclosed in class Abc.

```
1 public class Abc {
2     public interface Xyz {
3         void callback();
4     }
5     public static void registerCallback(Xyz xyz) {
6         // Implementation goes here
7     }
8 }
9 // Client Code
10 Abc.registerCallback(new Abc.Xyz() {
11     @Override
12     public void callback() {
13         // Custom callback implementation
14     }
15 });
```

Any code that cannot access Abc cannot access interface Xyz either. So the purpose of declaring an inner interface is to restrict its access from the outside world.

121. What is the meaning of Immutable in the context of String class in Java?

An Immutable object cannot be modified or changed in Java. String is an Immutable class in Java. Once a String object is created, it cannot be changed. When we assign the String to a new value, a new object is created.

122. Why is a String object considered immutable in Java?

Java language uses String for a variety of purposes. For this reason, it has marked String as Immutable. There is a concept of String literal in Java. Let's say there are 2 String variables A and B that reference a String object "TestData." Both variables refer to the same String literal. If one reference variable A changes the value of the String literal from "TestData" to "RealData," then it would affect the other variable as well. To avoid this, String is considered Immutable. In this case, if

variable A changes the value to "RealData," then a new String literal with "RealData" is created and A will point to the new String literal, while B will keep pointing to "TestData."

123. How many objects does following code create?

Code:

```
String s1="HelloWorld";  
String s2=" HelloWorld ";  
String s3=" HelloWorld ";
```

The above code creates only one object. Since there is only one String Literal "HelloWorld" created, all the references point to the same object.

124. How many ways are there in Java to create a String object?

Java provides two ways to create a String object. One is by using a String Literal, the other is by using the new operator.

125. How many objects does the following code create?

Code: `String s = new String("HelloWorld");` The above code creates two objects. One object is created in the String constant pool and the other is created on the heap in the non-pool area.

126. What is String interning?

String interning refers to the concept of using only one copy of a distinct String value that is immutable. It provides the advantage of making String processing efficient in both time and space complexity, but it introduces extra time during the creation of a String.

127. Why does Java use the String literal concept?

Java uses the String literal concept to make memory usage more efficient. If the same String already exists in the String constant pool, it can be reused, which saves memory.

128. What is the basic difference between a String and a StringBuffer object?

String is an immutable object; its value cannot change after creation. StringBuffer is a mutable object, allowing you to append or modify its contents in Java.

129. How do you create an immutable class in Java?

In Java, you can declare a class as final to make it immutable. The detailed steps are:

- Add the final modifier to the class to prevent it from being extended.
- Add the private modifier to all fields to prevent direct access.
- Do not provide any setter methods for member variables.
- Add the final modifier to all mutable fields to assign their value only once.
- Use deep copy to initialize all fields via the constructor.
- In the clone method, return a copy of the object instead of the actual object reference.

130. What is the use of the toString() method in Java?

In Java, the Object class has a toString() method that returns the string representation of an object. When you print an object, Java implicitly calls the toString() method. Java provides a default implementation, but you can override this method to return a custom format.

131. Arrange the three classes String, StringBuffer, and StringBuilder in order of efficiency for string processing operations?

StringBuilder is the most efficient class because it does not have the overhead of synchronization. StringBuffer is synchronized and has better performance than String, but it is slower than StringBuilder. String is the slowest for any string processing operations since each modification creates a new String literal. The decreasing order of efficiency is: StringBuilder, StringBuffer, String.

132. What is Exception Handling in Java?

Java provides an exception handling mechanism to handle runtime errors that occur in the JVM. There are checked exceptions in a program that we expect in certain situations, and the exception handling mechanism catches these exceptions and takes relevant actions.

133. In Java, what are the differences between a Checked and Unchecked Exception?

Checked exceptions extend the Throwable class but do not extend RuntimeException or Error classes. Unchecked exceptions extend the RuntimeException class. Checked exceptions are checked at compile time in Java, while unchecked exceptions occur at runtime and are not checked at compile time. Examples of checked exceptions include IOException and SQLException; examples of unchecked exceptions include NullPointerException and ArithmeticException.

134. What is the base class for Error and Exception classes in Java?

Both Error and Exception classes are derived from the Throwable class in Java.

135. What is a finally block in Java?

Java provides a finally block with a try block. This block is optional but always executed after the try block completes.

136. What is the use of a finally block in Java?

According to Java specifications, a finally block is always executed, whether an error occurs or not and whether an exception is handled or not. It is useful for performing cleanup tasks such as rolling back transactions, closing connections, or closing files.

137. Can we create a finally block without creating a catch block?

Yes, a finally block can follow a try block or a catch block. You can define a finally block directly after a try block.

138. Do we have to always put a catch block after a try block?

Java does not require a catch block after a try block. You can write a catch block or a finally block after a try block. Any exception you want to catch should be mentioned in the catch block.

139. In what scenarios will a finally block not be executed?

There are two main scenarios in which a finally block is not executed:

- The program exits by calling System.exit().
- A fatal error causes the JVM to crash.

140. Can we re-throw an Exception in Java?

Yes, Java allows you to re-throw an exception.

141. What is the difference between throw and throws in Java?

Java provides the throw keyword to throw an exception from a method or static block, and the throws keyword to declare probable exceptions thrown by a method in its declaration. Use throw to explicitly throw an exception and throws to declare exceptions in a method definition. You cannot propagate checked exceptions with throw only, but you can with throws. The throw call is followed by an instance, while the throws keyword is followed by a class or exception. You can throw only one exception at a time but mention multiple exceptions in a throws clause.

142. What is the concept of Exception Propagation?

In exception propagation, uncaught exceptions are passed up the call stack until the stack becomes empty. For example, if D() throws an exception and it is not caught, the exception will propagate from D to C to B to A unless one of the methods catches it.

143. When we override a method in a child class, can we throw an additional Exception that is not thrown by the parent class method?

Yes, Java allows you to throw an additional exception in a child class, but the additional exception must be an unchecked exception (such as RuntimeException).

144. What is the difference between Collection and Collections Framework in Java?

In Java, a Collection is an object that contains multiple elements of the same type and allows access to them as a single unit. The Java Collections Framework is a library that provides a common architecture for creating, updating, and accessing different types of collections, along with common methods frequently used by developers.

145. What are the main benefits of Collections Framework in Java?

Main benefits of the Collections Framework in Java include:

- Reusability: Provides common classes and utility methods usable with different types of collections, promoting code reuse.
- Quality: Improves program quality since the code is already tested and widely used.
- Speed: Increases development speed as programmers can focus on core logic and use generic collections provided by the framework.
- Maintenance: Open source code and widely available API documentation make it easy to maintain code written with the Collections Framework; one developer can easily work with code written by another.

146. What is the root interface of Collection hierarchy in Java?

The root interface of the Collection hierarchy in Java is the Collection interface. But the Collection interface extends the Iterable interface. Due to this, some people consider the Iterable interface as the root interface. The Iterable interface is present in the java.lang package, but the Collection interface is present in the java.util package. Oracle Java API docs mention that the Collection interface is a member of the Java Collections Framework, whereas the Iterable interface is not stated as a part of the Java Collections Framework in the Java docs. Due to this, the Collection interface is considered the root of the Collections Framework.

147. What are the main differences between Collection and Collections?

Main differences between Collection and Collections are as follows:

- Collection is an interface in Java. Collections is a class in Java.
- Collection is a base interface. Collections is a utility class in Java.
- Collection defines methods that are used for data structures that contain the objects. Collections defines the methods that are used for operations like access, find, etc., on a Collection.

148. What are the Thread-safe classes in Java Collections framework?

The Thread-safe classes in Java Collections Framework are:

- Stack
- Properties
- Vector
- Hashtable
- BlockingQueue
- ConcurrentMap
- ConcurrentNavigableMap

149. How will you efficiently remove elements while iterating a Collection?

The right way to remove elements from a collection while iterating is by using the `ListIterator.remove()` method. For example:

```
1 ListIterator<String> iter = myList.listIterator();
2 while (iter.hasNext()) {
3     iter.next();           // Move to next element
4     iter.remove();        // Remove the element just returned by next()
5 }
6
```

```

1 Iterator<String> iter = myList.iterator();
2 while (iter.hasNext()) {
3     iter.next();      // Must call next() before remove()
4     iter.remove();
5 }

```

Incorrect Way (causes IllegalStateException):

```

1 Iterator<String> iter = myList.iterator();
2 while (iter.hasNext()) {
3     iter.remove();    // Wrong: remove() called without next()
4 }
5

```

By doing so, we get `ConcurrentModificationException`. An iterator is first created to traverse the list, but at the same time the list is changed by the `remove()` method. In Java, it is not allowed for a thread to modify a collection while another thread is iterating it. `ListIterator` provides the capability of removing an object during traversal.

150. How will you convert a List into an array of integers like `int[]`?

We can use the `ArrayUtils` class in the Apache Commons Lang library.

```

1 int[] intArray = ArrayUtils.toPrimitive(myList.toArray(new Integer[0]));

```

This converts a `List<Integer>` to a primitive `int[]` using Apache Commons Lang.

```

1 int[] intArray = new int[myList.size()];
2 for (int i = 0; i < myList.size(); i++) {
3     intArray[i] = myList.get(i);
4 }

```

151. How will you convert an array of primitive integers `int[]` to a List collection?

We can use `ArrayUtils` in the Apache Commons Lang library for this purpose. Sample code is:

```

// Convert int[] to List<Integer>
List<Integer> intList = Arrays.asList(ArrayUtils.toObject(intArray));

```

Converts a primitive `int[]` to `Integer[]` using `ArrayUtils.toObject()`, then wraps it in a `List` using `Arrays.asList()`.

```

1 int[] intArray = {10, 20, 30};
2 List<Integer> intList = new ArrayList<>();
3
4 for (int i : intArray) {
5     intList.add(i);
6 }

```

152. How will you run a filter on a Collection?

We can use `CollectionUtils` of Apache for this purpose. We need to create a `Predicate` that defines the condition for our filter and then apply this `Predicate` in the `filter()` method.

```

1 List namesList = asList("Red", "Blue", "Green");
2 List shortNamesList = new ArrayList ();
3 shortNamesList.addAll(namesList);
4 CollectionUtils.filter(shortNamesList, new Predicate() {
5     public boolean evaluate(Object input) {
6         return ((String) input).length() < 5;
7     }
8 });

```

In this example, we filter any names that are less than 5 characters long. We can also use the Google Guava library for this. In Java 8, we can use Predicate to filter a Collection through Stream.

153. How will you convert a List to a Set?

There are two ways to convert a List to a Set in Java.

Option 1: Use HashSet. In this case we put a list into a HashSet. Internally, the hashCode() method is used to identify duplicate elements.

```
3 Set mySet = new HashSet (myList);
```

Option 2: Use TreeSet. In this case, we use our own comparator to find duplicate objects.

```
Set mySet = new TreeSet (myComparator);  
mySet.addAll(myList);
```

154. How will you remove duplicate elements from an ArrayList?

The trick in this question is to use a collection that does not allow duplicate elements. So we use a Set for this purpose.

Option 1: Use Set

If ordering of elements is not important then we just put the elements of ArrayList in a HashSet and then add them back to the ArrayList.

```
1 ArrayList myList = // ArrayList with duplicate elements  
2 Set mySet = new HashSet (myList);  
3 myList.clear();  
4 myList.addAll(mySet);
```

Option 2: Use LinkedHashSet

If ordering of elements is important then we put the elements of ArrayList in a LinkedHashSet and then add them back to the ArrayList.

```
1 ArrayList myList = // ArrayList with duplicate elements  
2 Set mySet = new LinkedHashSet (myList);  
3 myList.clear();  
4 myList.addAll(mySet);
```

155. How can you maintain a Collection with elements in sorted order?

In Java, there are many ways to maintain a Collection with elements in sorted order. Some collections like TreeSet store elements in the natural ordering. In case of natural ordering, we have to implement the Comparable interface for comparing the elements. We can also maintain custom ordering by providing a custom Comparator to a Collection. Another option is to use the utility method Collections.sort() to sort a List. This sorting gives $n\log(n)$ order of performance. But if we have to use this method multiple times then it will be costly on performance. Another option is to use a PriorityQueue that provides an ordered queue. The main difference between PriorityQueue and Collections.sort() is that PriorityQueue maintains a queue in order all the time, but we can only retrieve the head element from the queue. We cannot access the elements of PriorityQueue in random order. We can use TreeSet to maintain sorted order of elements in a collection if there are no duplicate elements in the collection.

156. What are the differences between the two data structures: a Vector and an ArrayList?

An ArrayList is a newer class than a Vector. A Vector is considered a legacy class in Java. The differences are:

- Synchronization: Vector is synchronized, but the ArrayList is not synchronized. So an ArrayList has faster operations than a Vector.
- Data Growth: Internally both an ArrayList and Vector use an array to store data. When an ArrayList is almost full it increases its size by 50% of the array size. Whereas a Vector increases it by doubling the underlying array size.

157. What are the differences between Collection and Collections in Java?

Main differences between Collection and Collections are:

- Type: Collection is an interface in Java. Collections is a class.
 - Features: Collection interface provides basic features of data structure to List, Set and Queue interfaces. Collections is a utility class to sort and synchronize collection elements. It has polymorphic algorithms to operate on collections.
 - Method Type: Most of the methods in Collection are at instance level. Collections class has mainly static methods that can work on an instance of Collection.
-

158. In which scenario, LinkedList is better than ArrayList in Java?

ArrayList is more popular than LinkedList in Java due to its ease of use and random access to elements feature. But LinkedList is better in the scenario when we do not need random access to elements or there are a lot of insertion, deletion of elements.

159. What are the differences between a List and Set collection in Java?

Main differences between a List and a Set are:

- Order: List collection is an ordered sequence of elements. A Set is just a distinct collection of elements that is unordered.
 - Positional Access: When we use a List, we can specify where exactly we want to insert an element. In a Set there is no order, so we can insert element anywhere without worrying about order.
 - Duplicate: In a List we can store duplicate elements. A Set can hold only unique elements.
-

160. What are the differences between a HashSet and TreeSet collection in Java?

Main differences between a HashSet and TreeSet are:

- Ordering: In a HashSet elements are stored in a random order. In a TreeSet, elements are stored according to natural ordering.
 - Null Value Element: We can store null value object in a HashSet. A TreeSet does not allow to add a null value object.
 - Performance: HashSet performs basic operations like add(), remove(), contains(), size() etc in a constant time. A TreeSet performs these operations at the order of log(n) time.
 - Speed: A HashSet is better than a TreeSet in performance for most of operations like add(), remove(), contains(), size() etc.
 - Internal Structure: A HashMap in Java internally backs a HashSet. A NavigableMap backs a TreeSet internally.
 - Features: A TreeSet has more features compared to a HashSet. It has methods like pollFirst(), pollLast(), first(), last(), ceiling(), lower() etc.
 - Element Comparison: A HashSet uses equals() method for comparison. A TreeSet uses compareTo() method for comparison to maintain ordering of elements.
-

161. In Java, how will you decide when to use a List, Set or a Map collection?

- If we want a Collection that does not store duplicate values, then we use a Set based collection.
- If we want to frequently access elements operations based on an index value then we use a List based collection. E.g. ArrayList
- If we want to maintain the insertion order of elements in a collection, then we use a List based collection.
- For fast search operation based on a key, value pair, we use a HashMap based collection.
- If we want to maintain the elements in a sorted order, then we use a TreeSet based collection.

162. What are the differences between a HashMap and a Hashtable in Java?

Main differences between a HashMap and a Hashtable are:

- Synchronization: HashMap is not a synchronized collection. If it is used in multi-thread environment, it may not provide thread safety. A Hashtable is a synchronized collection. Not more than one thread can access a Hashtable at a given moment of time. The thread that works on Hashtable acquires a lock on it and it makes other threads wait till its work is completed.
- Null values: A HashMap allows only one null key and any number of null values. A Hashtable does not allow null keys and null values.
- Ordering: A HashMap implementation by LinkedHashMap maintains the insertion order of elements. A TreeMap sorts the mappings based on the ascending order of keys. On the other hand, a Hashtable does not provide guarantee of any kind of order of elements. It does not maintain the mappings of key values in any specific order.
- Legacy: Hashtable was not the initial part of collection framework in Java. It has been made a collection framework member, after being retrofitted to implement the Map interface. A HashMap implements Map interface and is a part of collection framework since the beginning.
- Iterator: The Iterator of HashMap is a fail-fast and it throws ConcurrentModificationException if any other Thread modifies the map by inserting or removing any element except iterator's own remove() method. Enumerator of the Hashtable is not fail-fast.

163. What are the differences between a HashMap and a TreeMap?

Main differences between a HashMap and a TreeMap in Java are:

- Order: A HashMap does not maintain any order of its keys. In a HashMap there is no guarantee that the element inserted first will be retrieved first.
- In a TreeMap elements are stored according to natural ordering of elements. A TreeMap uses compareTo() method to store elements in a natural order.
- Internal Implementation: A HashMap uses Hashing internally. A TreeMap internally uses Red-Black tree implementation.
- Parent Interfaces: A HashMap implements Map interface. TreeMap implements NavigableMap interface.
- Null values: A HashMap can store one null key and multiple null values. A TreeMap can not contain null key but it may contain multiple null values.
- Performance: A HashMap gives constant time performance for operations like get() and put(). A TreeMap gives order of log(n) time performance for get() and put() methods.
- Comparison: A HashMap uses equals() method to compare keys. A TreeMap uses compareTo() method for maintaining natural ordering.
- Features: A TreeMap has more features than a HashMap. It has methods like pollFirstEntry(), pollLastEntry(), tailMap(), firstKey(), lastKey() etc. that are not provided by a HashMap.

164. What are the differences between Comparable and Comparator?

Main differences between Comparable and Comparator are:

- Type: Comparable is an interface in Java where T is the type of objects that this object may be compared to.
- Comparator is also an interface where T is the type of objects that may be compared by this comparator.
- Sorting: In Comparable, we can only create one sort sequence. In Comparator we can create multiple sort sequences.

- Method Used: Comparator interface in Java has method `public int compare (Object o1, Object o2)` that returns a negative integer, zero, or a positive integer when the object o1 is less than, equal to, or greater than the object o2. A Comparable interface has method `public int compareTo(Object o)` that returns a negative integer, zero, or a positive integer when this object is less than, equal to, or greater than the object o.
 - Objects for Comparison: The Comparator compares two objects given to it as input. Comparable interface compares "this" reference with the object given as input.
 - Package location: Comparable interface in Java is defined in `java.lang` package. Comparator interface in Java is defined in `java.util` package.
-

165. In Java, what is the purpose of Properties file?

A Properties file in Java is a list of key-value pairs that can be parsed by the `java.util.Properties` class. Generally, a Properties file has the extension `.properties`, e.g. `myapp.properties`. Properties files are used for many purposes in all kinds of Java applications, such as storing configuration, initial data, application options, etc. When we change the value of a key in a properties file, there is no need to recompile the Java application, providing the benefit of changing values at runtime.

166. What is the reason for overriding equals() method?

The `equals()` method in the `Object` class is used to check whether two objects are the same or not. If we want a custom implementation, we can override this method. For example, a `Person` class has a first name, last name, and age. If we want two `Person` objects to be equal based on name and age, then we can override `equals()` to compare the first name, last name, and age of `Person` objects. Generally, in `HashMap` implementation, if we want to use an object as a key, then we override the `equals()` method.

167. How does hashCode() method work in Java?

The `Object` class in Java has the `hashCode()` method, which returns a hash code value as an integer. The `hashCode()` is a native method, and its implementation is not pure Java. Java doesn't generate `hashCode()`; however, `Object` generates a hash code based on the memory address of the instance of the object. If two objects are the same, then their `hashCode()` is also the same.

168. Is it a good idea to use Generics in collections?

Yes. A collection is a group of elements put together in an order or based on a property. Often the type of element can vary, but the properties and behavior of a `Collection` remain the same. Therefore, it is good to create a `Collection` with Generics so that it is type-safe and can be used with a wide variety of elements.

169. What is the difference between Collections.emptyList() and creating new instance of Collection?

In both approaches, we get an empty list. But `Collections.emptyList()` returns an immutable list. We cannot add new elements to an immutable empty list. `Collections.emptyList()` works like the Singleton pattern; it does not create a new instance of `List` but reuses an existing empty list instance. Therefore, `Collections.emptyList()` gives better performance if we need to get an empty list multiple times.

170. How will you copy elements from a Source List to another list?

There are two options to copy a Source List to another list.

Option 1: Use `ArrayList` constructor

```
ArrayList newList = new ArrayList(sourceList);
```

Option 2: Use Collections.copy() To use Collections.copy(), the destination list should be of the same or larger size than the source list.

```
ArrayList newList = new ArrayList(sourceList.size());
```

```
Collections.copy(newList, sourceList);
```

Collections.copy() does not reallocate the capacity of the destination list if it does not have enough space to contain all elements of the source list. It throws IndexOutOfBoundsException. The benefit of Collections.copy() is that it guarantees that the copy will happen in linear time. It is also good for scenarios when we want to reuse an array instead of allocating more memory in the constructor of ArrayList. One limitation of Collections.copy() is that it can accept only List as source and destination parameters.

171. What are the Java Collection classes that implement List interface?

- ArrayList
- AbstractSequentialList
- ArrayList
- AttributeList
- CopyOnWriteArrayList
- LinkedList
- RoleList
- RoleUnresolvedList
- Stack
- Vector

172. What are the Java Collection classes that implement Set interface?

- AbstractSet
- ConcurrentSkipListSet
- CopyOnWriteArraySet
- EnumSet
- HashSet
- JobStateReasons
- LinkedHashSet
- TreeSet

173. What is the difference between an Iterator and ListIterator in Java?

- ListIterator can be used to traverse only a List, but Iterator can be used to traverse List, Set, Queue, etc.
- An Iterator traverses the elements in one direction only. ListIterator can traverse elements in both backward and forward directions.
- Iterator cannot provide the index of an element in the data structure. ListIterator provides methods like nextIndex() and previousIndex() to get the index of an element during traversal.
- Iterator does not allow us to add an element to the collection while traversing it; it throws ConcurrentModificationException. ListIterator allows adding an element at any point while traversing a list.
- An existing element's value cannot be replaced by using Iterator. ListIterator provides the method set(e) to replace the value of the last element returned by next() or previous() methods.

174. What is the difference between Iterator and Enumeration?

- Enumeration is an older interface; Iterator is a newer interface.
- Enumeration can only traverse legacy collections; Iterator can traverse both legacy and newer collections.
- Enumeration does not provide the `remove()` method, so we cannot remove any element during traversal. Iterator provides the `remove()` method.
- Iterator is a fail-fast interface; it gives `ConcurrentModificationException` if any thread tries to modify an element in the collection being iterated. Enumeration is not fail-fast.
- Method names in Iterator are shorter than in Enumeration.

175. What is the difference between an ArrayList and a LinkedList data structure?

- Data Structure: An ArrayList is an indexed-based dynamic array. A LinkedList is a doubly linked list data structure.
- Insertion: It is easier to insert new elements in a LinkedList since there is no need to resize an array. Insertion in ArrayList is $O(n)$ since it may require resizing of the array and copying its contents to a new array.
- Remove elements: LinkedList has better performance in removal of elements than ArrayList.
- Memory Usage: LinkedList uses more memory than ArrayList since it has to maintain links for next and previous nodes as well.
- Access: LinkedList is slower in accessing an element since we have to traverse the list one by one to access the right location.

176. What is the difference between a Set and a Map in Java?

- Duplicate Elements: A Set does not allow inserting duplicate elements. A Map does not allow using duplicate keys, but it allows inserting duplicate values for unique keys.
- Null values: A Set allows inserting a maximum of one null value. In a Map, we can have a single null key at most and any number of null values.
- Ordering: A Set does not maintain any order of elements. Some subclasses of a Set can sort the elements in an order like `LinkedHashSet`. A Map does not maintain any order of its elements. Some of its subclasses like `TreeMap` store elements of the map in ascending order of keys.

177. What is the use of a Dictionary class?

The Dictionary class in Java is used to store key-value pairs. Any non-null object can be used for a key or value, but we cannot insert a null key or null object in Dictionary. The Dictionary class is deprecated now and should not be used in newer implementations.

178. What is the default size of load factor in a HashMap collection in Java?

The default value of load factor in a HashMap is 0.75.

179. What is the significance of load factor in a HashMap in Java?

A HashMap in Java has a default initial capacity of 16 and the load factor is 0.75f (i.e. 75% of current map size). The load factor of a HashMap is the level at which its capacity should be doubled. For example, in a HashMap of capacity 16 and load factor .75, the capacity will become 32 when the HashMap is 75% full. Therefore, after storing the 12th key-value pair ($16 * .75 = 12$) into HashMap, its capacity becomes 32.

180. What are the major differences between a HashSet and a HashMap?

- Base class: A HashSet class implements the Set interface, whereas a HashMap class implements the Map interface.
- Storage: A HashSet is used to store distinct objects. A HashMap is used for storing key & value pairs, so that these can be retrieved by key later on.
- Duplicate Elements: A HashSet does not allow storing duplicate elements. A HashMap also does not allow duplicate keys, but we can store duplicate values in a HashMap.
- Null Elements: In a HashSet, we can store a single null value. In a HashMap, we can store a single null key, but any number of null values.
- Element Type: A HashSet contains only values of objects as its elements, whereas a HashMap contains entries (key-value pairs).
- Iteration: By using an Iterator, we can iterate a HashSet, but a HashMap has to be converted into a Set for iteration.

181. What are the similarities between a HashSet and a HashMap in Java?

As the name suggests, HashSet and HashMap are hashing-based collections. Similarities between HashSet and HashMap are:

- Thread Safety: Both HashMap and HashSet are not synchronized collections. Therefore, they are not good for thread-safe operations. To make these thread-safe, we need to explicitly use synchronized versions.
- Order of Elements: None of these classes guarantee the order of elements. These are unordered collections.
- Internal Implementation: A HashMap backs up a HashSet internally. So HashSet uses a HashMap for performing its operations.
- Performance: Both of these collections provide constant time performance for basic operations such as insertion and removal of elements.

182. What is the reason for overriding equals() method?

The equals() method in the Object class is used to check whether two objects are the same or not. If we want a custom implementation, we can override this method. For example, a Person class has first name, last name, and age. If we want two Person objects to be equal based on name and age, then we can override equals() method to compare the first name, last name, and age of Person objects. Generally, in HashMap implementation, if we want to use an object as a key, then we override equals() method.

183. How can we synchronize the elements of a List, a Set or a Map?

Sometimes we need to make collections thread-safe for use in a multi-threading environment. In Java, the Collections class provides useful static methods to make a List, Set, or Map as synchronized collections. Some of these methods are:

- static Collection synchronizedCollection(Collection c) Returns a synchronized (thread-safe) collection backed by the specified collection.
- static List synchronizedList(List list) Returns a synchronized (thread-safe) list backed by the specified list.
- static Map synchronizedMap(Map m) Returns a synchronized (thread-safe) map backed by the specified map.
- static Set synchronizedSet(Set s) Returns a synchronized (thread-safe) set backed by the specified set.
- static SortedMap synchronizedSortedMap(SortedMap m) Returns a synchronized (thread-safe) sorted map backed by the specified sorted map.
- static SortedSet synchronizedSortedSet(SortedSet s) Returns a synchronized (thread-safe) sorted set backed by the specified sorted set.

184. What is Hash Collision? How does Java handle hash-collision in HashMap?

In a hashing scenario, at times two different objects may have the same hashCode but they may not be equal. Therefore, Java will face an issue while storing two different objects with the same hashCode in a HashMap. This kind of situation is called Hash Collision. There are different techniques for resolving or avoiding Hash Collision. But in HashMap, Java simply replaces the object at the old key with the new object in case of Hash Collision.

185. What are the Hash Collision resolution techniques?

To resolve a Hash Collision, we can use one of the following techniques:

- Separate Chaining with Linked List
- Separate Chaining with List Head Cells
- Open Addressing with Coalesced Hashing
- Open Addressing with Cuckoo Hashing
- Hopscotch Hashing
- Robinhood Hashing

186. What is the difference between Queue and Stack data structures?

Queue is a FIFO data structure. FIFO stands for First In First Out. It means the element added first will be removed first from the queue. A real-world example of a Queue is a line for buying tickets at a station. The person entering first in the queue is served first. Stack is a LIFO data structure. LIFO stands for Last In First Out. The element that is added last is removed first from the collection. In a Stack, elements are added or removed from the top of the stack. A real-world example of a Stack is the back button in a browser, where you can go back one by one in reverse order of adding webpages to history.

187. What is an Iterator in Java?

Iterator is an interface in Java to access the elements in a collection, available in the java.util package. It provides methods to iterate over a Collection class in Java. Iterator interface in Java is based on the Iterator design pattern. By using an Iterator, one can traverse a container of objects and can also access the objects in the container. A container of objects is a Collection class in Java.

188. What is the difference between Iterator and Enumeration in Java?

Main differences between Iterator and Enumeration in Java are:

- Version: Enumeration interface has been in Java since JDK 1.0. Iterator interface was introduced in Java 1.2.
- remove() method: The main difference between Enumeration and Iterator interface is the remove() method. Enumeration can just traverse a Collection object. If we use Enumeration, we cannot do any modifications to a Collection while traversing the collection. Iterator interface provides remove() method to remove an element while traversing the Collection. There is no remove() method in Enumeration interface.
- Method names: Names of methods in Iterator interface are hasNext(), next(), remove(). Names of methods in Enumeration interface are hasMoreElements(), nextElement().
- Legacy Interface: Enumeration is considered as a legacy interface. It is used to traverse legacy classes like Vector, Stack, and Hashtable. Iterator is a newer interface that is used to traverse almost all of the classes in the Java Collections framework.

- Fail-fast vs. Fail-safe: Iterator is based on the fail-fast principle. It throws `ConcurrentModificationException` if a collection is modified during iteration. Enumeration is based on the fail-safe principle. It doesn't throw any exception if a collection is modified during traversal.
- Safety: Since Iterator is fail-fast and does not allow modification of a collection by other threads, it is considered safer than Enumeration.

189. What is the design pattern used in the implementation of Enumeration in Java?

Enumeration is based on the Iterator design pattern. The Iterator design pattern provides a common interface with methods to traverse the collection of objects. It hides the underlying implementation details of the collection.

190. Which methods do we need to override to use an object as key in a HashMap?

If we want to use an object as a key in a HashMap in Java, then we have to make sure that it has the implementation of `equals()` and `hashCode()` methods.

191. How will you reverse a List in Java?

In the Collections class, Java provides a method `reverse(List list)` that can be used to reverse a List.

Java

```
Collections.reverse(myList);
```

Show more lines

192. How will you convert an array of String objects into a List?

Java provides the Arrays class in the `java.util` package. Arrays class has a method `asList()` that accepts an array as input and returns a List as output.

```
1 public static List asList(T... a)
2 String[] myArray = {"George", "Jack", "Ryan"};
3 List myList = Arrays.asList(myArray);
```

193. What is the difference between peek(), poll(), and remove() methods of Queue interface in Java?

In a Java Queue, `poll()` and `remove()` methods can be used for removing the head object of Queue. The main difference arises in the case when the Queue is empty. If the Queue is empty, then `poll()` method returns a null value. If the Queue is empty, then `remove()` method throws `NoSuchElementException`. In a Java Queue, `peek()` method retrieves the head of Queue but does not remove it. If the Queue is empty, then `peek()` method returns a null value.

194. What is the difference between Array and ArrayList in Java?

The main differences between Array and ArrayList in Java are:

- Size: Array in Java is fixed in size. We cannot change the size of an array after creating it. ArrayList is dynamic in size. When we add elements to an ArrayList, its capacity increases automatically.
- Performance: In Java, Array and ArrayList give different performance for different operations.
- `add()` or `get()`: Adding an element to or retrieving an element from an array or ArrayList object has similar performance. These are constant time operations.
- `resize()`: Automatic resize of ArrayList slows down the performance. ArrayList is internally backed by an Array. In `resize()`, a temporary array is used to copy elements from the old array to the new array.

- Primitives: Array can contain both primitive data types as well as objects. But ArrayList cannot contain primitive data types. It contains only objects.
 - Iterator: In an ArrayList, we use an Iterator object to traverse the elements. We use a for loop for iterating elements in an array.
 - Type Safety: Java helps in ensuring type safety of elements in an ArrayList by using Generics. An Array can contain objects of the same type of class. If we try to store a different data type object in an Array, then it throws **ArrayStoreException**.
 - Length: Size of ArrayList can be obtained by using size() method. Every array object has a length variable that is the same as the length/size of the array.
 - Adding elements: In an ArrayList, we can use add() method to add objects. In an Array, assignment operator is used for adding elements.
 - Multi-dimension: An Array can be multi-dimensional. An ArrayList is always of single dimension.
-

195. How will you insert, delete, and retrieve elements from a HashMap collection in Java?

We use the following methods to insert, delete, and retrieve elements in a HashMap.

- Retrieve: We use get() method to retrieve elements from a HashMap.
 - Value get(Object key)
 - Insert: We use put() method to insert a key-value pair in a HashMap.
 - Value put(Key k, Value v)
 - Delete: We use remove() method to delete a key-value pair from the HashMap.
 - Value remove(Object key)
-

196. What are the main differences between HashMap and ConcurrentHashMap in Java?

Main differences between HashMap and ConcurrentHashMap are:

- Synchronization: A HashMap is not synchronized. But a ConcurrentHashMap is a synchronized object.
 - Null Key: A HashMap can have one null key and any number of null values. A ConcurrentHashMap cannot have null keys or null values.
 - Multi-threading: A ConcurrentHashMap works well in a multi-threading environment.
-

197. What is the increasing order of performance for the following collection classes in Java?

The increasing order of performance is:

- Hashtable
- Collections.SynchronizedMap
- ConcurrentHashMap
- HashMap

Hashtable has the worst performance and HashMap has the best performance.

198. Why does Map interface not extend Collection interface in Java?

A Map is a collection of objects. But the Map interface is not compatible with the Collection interface in Java. A Map requires a key as well as a value, so it requires two parameters to add an element to a HashMap. But the Collection interface provides add(Object o) method with only one parameter. Map collection has to provide methods like valueSet, keySet, etc.

These methods are specific to Map collection, whereas methods in the Collection interface can be reused by List, Set, Queue, etc.

199. What are the different ways to iterate elements of a list in Java?

There are mainly two ways to iterate the elements of a list in Java:

- Iterator: We can get an Iterator for a list and use it to iterate the objects of the list.
 - For-each loop: We can use a for-each loop to traverse all the elements of a list.
-

200. What is CopyOnWriteArrayList? How is it different from ArrayList in Java?

CopyOnWriteArrayList was introduced in Java 5. It is a thread-safe collection, similar to an ArrayList. In CopyOnWriteArrayList, all mutative operations (add, set, etc.) are implemented by making a fresh copy of the underlying array. The iterator of CopyOnWriteArrayList is guaranteed not to throw ConcurrentModificationException, but it also does not reflect any additions or removals that happened to the list after the iterator was created. All elements, including null, are permitted in CopyOnWriteArrayList.

201. How is the remove() method implemented in a HashMap?

The remove() method in HashMap uses logic similar to the get() method. First, it locates the correct bucket in the HashMap for an entry, then removes the element e within that bucket. It is similar to removing a node from a single-linked list. If e is the first element in the bucket, the corresponding element of Hash is set to e.next. Otherwise, the next field of the element just before e is set to e.next.

202. What is BlockingQueue in Java Collections?

BlockingQueue was introduced in Java 1.5 and extends the Queue interface. BlockingQueue supports operations that wait for the queue to become non-empty when retrieving an element, and operations that wait for space to become available in the queue when storing an element. Some features of BlockingQueue are:

- It does not accept null elements.
- Its main use is in producer-consumer problems.
- BlockingQueue implementation is thread-safe.
- It can be used in inter-thread communications.
- It does not support any kind of "close" or "shutdown" operation to indicate that no more items will be added.

203. How is TreeMap class implemented in Java?

Internally, the TreeMap class in Java uses a Red-Black tree. It is a NavigableMap and sorts the keys in natural order or uses a Comparator supplied at creation time. The implementation of TreeMap is not synchronized in Java.

204. What is the difference between Fail-fast and Fail-safe iterator in Java?

Differences between Fail-fast and Fail-safe iterators are as follows:

- Fail-fast iterator throws ConcurrentModificationException, while Fail-safe iterator does not.
- Fail-fast iterator does not clone the original collection, but Fail-safe iterator creates a copy of the original collection of objects.
- A Fail-fast iterator immediately throws an exception when it encounters failure, whereas a Fail-safe Iterator works on a copy of the collection instead of the original collection.

205. How does ConcurrentHashMap work in Java?

ConcurrentHashMap extends AbstractMap and was introduced in Java 1.5. It provides concurrency in a collection based on a HashMap. All methods are thread-safe in ConcurrentHashMap. Internally, there is a Hashtable backing a ConcurrentHashMap, which supports concurrent methods for data retrieval and updates. It has the same functional

specification as a Hashtable. ConcurrentHashMap also supports a set of sequential and bulk operations that accept a parallelismThreshold argument.

206. What is the importance of hashCode() and equals() methods?

In a HashMap collection, it is very important for a key object to implement both hashCode() and equals() methods. If the hashCode() method returns the same hashcode for all key objects, the hash collision will be high in the HashMap. Also, with the same hashcode, the equals method will yield the same result, making the HashMap inefficient. The problem arises when HashMap treats both outputs as the same instead of different, overwriting the most recent key-value pair with the previous key-value pair. Therefore, it is important to implement hashCode() and equals() methods correctly for an efficient HashMap collection.

207. What is the contract of hashCode() and equals() methods in Java?

The contract of hashCode() and equals() methods in Java is as follows:

If object1.equals(object2), then object1.hashCode() == object2.hashCode() should always be true. This means if two objects are equal, their hashCode should be the same.

If object1.hashCode() == object2.hashCode() is true, it does not guarantee that object1.equals(object2). This means if two objects have the same hashCode, they can still have different values and may not be equal objects.

208. What is an EnumSet in Java?

EnumSet is a specialized implementation of Set. Its main features are:

- It is mainly used with enum types.
- All elements in an EnumSet must come from a single enum type when the set is created.
- Internally, EnumSet is represented as a bit vector.
- The iterator of EnumSet traverses the elements in their natural order (the order in which the enum constants are declared).
- Null elements are not permitted; inserting a null element throws NullPointerException.
- EnumSet is not a synchronized collection. For multi-threading scenarios, EnumSet should be synchronized.
- EnumSet is a good alternative to int-based "bit flags" implementation.

209. What are the main Concurrent Collection classes in Java?

Java 1.5 introduced the package java.util.concurrent, which contains thread-safe collection classes. These collections can be modified while iterating, and their iterators are fail-safe. Main Concurrent Collection classes in Java 8 are:

- ArrayBlockingQueue
- CopyOnWriteArrayList
- CopyOnWriteArraySet
- ConcurrentHashMap
- ConcurrentLinkedDeque
- ConcurrentLinkedQueue
- LinkedBlockingQueue
- LinkedBlockingDeque
- PriorityBlockingQueue

210. How will you convert a Collection to SynchronizedCollection in Java?

Java provides an easy method in the java.util.Collections class to create a thread-safe collection from a regular collection. We can use the synchronizedCollection() method for this purpose. For any class of type T, use the following method:

```
static Collection synchronizedCollection(Collection c)
```

211. How is IdentityHashMap different from a regular Map in Java?

IdentityHashMap in Java implements the Map interface but is not a general-purpose implementation. It violates the general contract of the Map interface by implementing equals() differently. In IdentityHashMap, two keys k1 and k2 are equal if and only if (k1 == k2). In a normal Map implementation (like HashMap), two keys k1 and k2 are considered equal if and only if (k1 == null ? k2 == null : k1.equals(k2)). It uses reference-equality in place of object-equality when comparing keys and values.

212. What is the main use of IdentityHashMap?

Main uses of IdentityHashMap are:

- Topology Preservation: Used in topology-preserving object graph transformations, such as serialization or deep-copying, where a "node table" must be maintained to track all object references already processed.
- The node table should not consider distinct objects as equal, even if they happen to be equal.
- Proxy objects: Used to maintain proxy objects, for example, in debugging programs where a proxy object is maintained for each object in the program being debugged.

213. How can we improve the performance of IdentityHashMap?

IdentityHashMap has one tuning parameter for performance improvement: expectedMaxSize. This parameter sets the maximum number of key-value mappings expected in the map and is used to determine the initial number of buckets in the hash table. If the number of key-value mappings exceeds this size, the number of buckets is increased (rehashing), which can be expensive. Therefore, it's better to create identity hash maps with a sufficiently large expected maximum size. Iteration over a Map requires time proportional to the number of buckets, so iteration may take extra time due to a large number of buckets. The expectedMaxSize value should be set with these aspects in mind.

214. Is IdentityHashMap thread-safe?

IdentityHashMap is not thread-safe because its methods are not synchronized. The iterators returned by the iterator method are fail-fast, but this behavior cannot be guaranteed. Since the iterator is fail-fast, it throws ConcurrentModificationException.

215. What is a WeakHashMap in Java?

WeakHashMap is similar to IdentityHashMap and is internally represented by a Hashtable. It is not a synchronized class, but can be made thread-safe using Collections.synchronizedMap(). An entry in WeakHashMap is automatically removed when it is no longer in ordinary use; the presence of a mapping for a given key does not prevent the key from being discarded by the garbage collector. WeakHashMap permits null keys and null values.

216. How can you make a Collection class read-only in Java?

In Java, useful methods exist to make a Collection class read-only:

- Collections.unmodifiableMap(Map m)
- Collections.unmodifiableList(List l)
- Collections.unmodifiableSet(Set s)
- Collections.unmodifiableCollection(Collection c)

217. When is UnsupportedOperationException thrown in Java?

In a Java collection, UnsupportedOperationException is thrown when the requested operation is not supported by the collection. It is an unchecked exception thrown on optional operations. If there are optional add() or remove() methods in a read-only collection, this exception can be thrown.

218. Let's say there is a Customer class. We add objects of Customer class to an ArrayList. How can we sort the Customer objects in ArrayList by using the customer firstName attribute?

There are two ways to handle this scenario:

- Comparable: Implement the Comparable interface for the Customer class and compare customer objects by the firstName attribute.

- **Comparator:** Implement Comparator for comparing two Customer objects based on the firstName attribute, then use this comparator object in the sort method of the Collections class.

219. What is the difference between Synchronized Collection and Concurrent Collection?

Java 1.5 introduced many concurrent collection classes such as ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue, etc. Java also provides utility methods to get a synchronized copy of collections like ArrayList and HashMap using Collections.synchronizedList() and Collections.synchronizedMap() methods. The main difference is performance: Concurrent collection classes have better performance than synchronized collections because they lock only a portion of the class to achieve concurrency and thread-safety.

220. What is the scenario to use ConcurrentHashMap in Java?

ConcurrentHashMap is more suitable for scenarios with multiple reader threads and one writer thread. In this case, the map is locked only during the write operation. If there is an equal number of reader and writer threads, ConcurrentHashMap performance is similar to a Hashtable or a synchronized HashMap.

221. How will you create an empty Map in Java?

There are two ways to create an empty Map in Java:

- **Immutable:** If an immutable empty Map is needed, use the following code:
- `myMap = Collections.emptyMap();`
- **Any map:** For other scenarios, use the following code:
- `myMap = new HashMap();`

222. What is the difference between remove() method of Collection and remove() method of Iterator?

In the Collection interface, remove(Object o) is used to remove objects from a Collection. The List interface also provides remove(int index) to remove an object at a specific index. These methods are used to remove an entry from a Collection while no thread is iterating over it. While iterating over a Collection, the remove() method of Iterator should be used, which removes the current element from the Iterator's point of view. Using remove() of Collection or List during iteration can result in ConcurrentModificationException. Therefore, it is recommended to use the remove() method of Iterator during traversal.

223. Between an Array and ArrayList, which one is the preferred collection for storing objects?

An ArrayList is backed by an array internally, and there are many usability advantages to using an ArrayList over an array in Java. Array has a fixed length at creation and cannot be resized. ArrayList is dynamic in size, and once it reaches a threshold, it automatically allocates a new array and copies the contents of the old array to the new one. ArrayList also supports Generics, but Array does not. For example, storing an Integer object in a String array at runtime will throw ArrayStoreException, whereas using ArrayList will result in a compile-time error, preventing runtime issues. If the size is known in advance and resizing is not needed, Array should be used instead of ArrayList.

224. Is it possible to replace Hashtable with ConcurrentHashMap in Java?

Yes, ConcurrentHashMap can replace Hashtable in Java, but it requires careful observation because the locking behavior differs. Hashtable locks the whole Map, while ConcurrentHashMap locks only parts of the Map. Compound operations like if(Hashtable.get(key) == null) put(key, value) work in Hashtable but not in ConcurrentHashMap. For such scenarios, use putIfAbsent() in ConcurrentHashMap.

225. How is CopyOnWriteArrayList different from ArrayList and Vector classes?

CopyOnWriteArrayList was introduced in Java 1.5 and implements the List interface. It provides better concurrent access methods than a synchronized List. In CopyOnWriteArrayList, concurrency is achieved by copying the ArrayList over each write and replacing the original instead of locking.

CopyOnWriteArrayList does not throw ConcurrentModificationException during iteration and is a thread-safe list. It differs from Vector in terms of concurrency, providing better concurrency by reducing contention among readers and writers.

226. Why does ListIterator have an add() method but Iterator does not?

ListIterator can iterate in both directions of a Collection and maintains pointers for previous and next elements. The add() method allows adding an element into the list immediately before the element returned by next(). A subsequent call to next() will not be affected, and a call to previous() will return the newly added element. Iterator only traverses in one direction, so there is no purpose for an add() method.

227. Why do we sometimes get ConcurrentModificationException during iteration?

ConcurrentModificationException occurs when an object is removed using the remove() method of a Collection or List while an Iterator is traversing it. If an Iterator detects any structural change in the Collection, it may throw this exception.

228. How will you convert a Map to a List in Java?

In Java, a Map has three collection sets: key set, value set, and key-value set. Each of these sets can be converted to a List using a constructor. Sample code:

```
List keyList = new ArrayList(map.keySet());  
List valueList = new ArrayList(map.values());  
List entryList = new ArrayList(map.entrySet());
```

229. How can we create a Map with reverse view and lookup in Java?

In a Map, a value can be looked up using a distinct key. In a Map with reverse view and lookup, the values must also be distinct, allowing one-to-one mapping between keys and values. With this constraint, a key can be looked up by its value. This data structure is called a bi-directional map. There is no built-in data structure similar to a reverse lookup Map in JDK, but Apache Commons Collections and Guava libraries provide implementations called BidiMap and BiMap, which enforce one-to-one mapping between keys and values.

230. How will you create a shallow copy of a Map?

In Java, most Map implementations provide a constructor to create a copy of another map, but the copy method is not synchronized. Therefore, another thread can modify the map during copying. To prevent this, use Collections.synchronizedMap() to first create a thread-safe map. Another way to create a shallow copy is by using the clone() method, though it is not a recommended approach.

231. Why we cannot create a generic array in Java?

Java does not allow creation of arrays with generics as elements. In Java, an array has to know the type information of its elements at runtime. This information is used at runtime to throw **ArrayStoreException** if the data type of an element to be inserted does not match the type of the array. In the case of generics, the type information of a collection is erased at runtime by Type Erasure. Due to this, arrays cannot use generics as elements.

232. What is a PriorityQueue in Java?

A PriorityQueue is a data structure based on Queue. Unlike Queue, the elements in a PriorityQueue are not returned in FIFO order. A PriorityQueue maintains the natural order of its elements or uses a Comparator provided at initialization. It is an unbounded queue based on a priority heap. PriorityQueue does not allow null values. We cannot add any object that does not provide natural ordering to a PriorityQueue. PriorityQueue in Java is not thread-safe. It provides O(log n) time for enqueueing and dequeueing operations.

233. What are the important points to remember while using Java Collections Framework?

- Interfaces: For Collections, we should write code with generic interfaces instead of concrete implementation. This maintains the flexibility to change the implementation later.
- Generics: We should use Generics for type-safety and to avoid ClassCastException at runtime.
- Collections: It is recommended to use the Collections utility class for algorithms and various other common methods for Collections.

- **Right Type:** We have to choose the right type of Java collection based on our need. If size is fixed, use Array over ArrayList. If we do not want duplicate elements, use Set. If we need the ability to iterate the elements of a Map in the order of insertion, use a TreeMap.
- **Initial Size:** In some collection classes, we can specify the initial size/capacity. Therefore, we should estimate the number of elements in a Collection before deciding the right collection type to avoid rehashing or resizing.
- **Map:** We should use immutable classes provided by Java as key elements in a Map.

234. How can we pass a Collection as an argument to a method and ensure that method will not be able to modify it?

To ensure that a method is not able to modify a Collection passed as an argument, we have to make the Collection read-only. We can make a read-only collection by using `Collections.unmodifiableCollection(Collection c)` method. This ensures that any operation to change the collection will throw `UnsupportedOperationException`.

235. Can you explain how HashMap works in Java?

In Java, a HashMap works on the concept of hashing. A HashMap stores both key and value objects in a bucket. It is stored as an Entry object that implements the `Map.Entry` interface. The key object used in a HashMap must provide implementations for `hashCode()` and `equals()` methods. When the `put()` method is used to store a key-value pair, the HashMap implementation calls `hashCode()` on the key object to calculate a hash, which is used to find the bucket where the Entry object will be stored. When the `get()` method is used to retrieve a value, the hash of the key object is calculated to find the bucket containing the key. If more than one key is stored in the same location, `equals()` is used to find the exact key object to retrieve the value.

236. Can you explain how HashSet is implemented in Java?

Internally, a HashSet uses a HashMap to store the elements and to maintain the uniqueness of elements. When we create a HashSet object, a corresponding HashMap object is also created. When we insert an element in HashSet, it is inserted into the corresponding HashMap.

237. What is a NavigableMap in Java?

As the name suggests, NavigableMap provides the capability to navigate the keys of a Map in Java. A NavigableMap extends the `SortedMap` interface. Some interesting methods of NavigableMap are `descendingKeySet()`, `descendingMap()`, `headMap()`, and `tailMap()`.

238. What is the difference between descendingKeySet() and descendingMap() methods of NavigableMap?

The `descendingKeySet()` method of NavigableMap returns a `NavigableSet` in which elements are stored in reversed order compared to the original key set. The returned view is internally represented by the original key set of NavigableMap, so any changes to the descending set are reflected in the original set. However, it is not recommended to remove elements directly from the key set; we should use the `Map.remove()` method. The `descendingMap()` method returns a NavigableMap that is an inverse view of the original map, with the order of the elements in reverse. Changes to this view are also reflected in the original map.

239. What is the advantage of NavigableMap over Map?

The main advantage of NavigableMap over Map is the navigation capability. It provides the capabilities of a Map, SortedMap, and navigation in one collection. It can return the closest matches for given search targets. Methods like `lowerEntry`, `floorEntry`, `ceilingEntry`, and `higherEntry` return `Map.Entry` objects associated with keys less than, less than or equal, greater than or equal, and greater than a given key, respectively. Methods like `lowerKey`, `floorKey`, `ceilingKey`, and `higherKey` return only the associated keys. All of these methods are designed for locating, not traversing, entries.

240. What is the difference between headMap(), tailMap() and subMap() methods of NavigableMap?

The `headMap()` method returns a view of the original NavigableMap that contains the elements that are less than a given element.

```
NavigableMap original = new TreeMap();
original.put("1", "1");
```

```
original.put("2", "2");
original.put("3", "3");
// this headmap1 will contain elements "1" and "2"
SortedMap headmap1 = original.headMap("3");
// this headmap2 will contain elements "1", "2", and "3" because "inclusive"=true
NavigableMap headmap2 = original.headMap("3", true);
```

The tailMap() method works similarly to headMap(), but returns all elements that are higher than the given input element. The subMap() method accepts two parameters demarcating the boundaries of the view map to return. All three methods return a subset of the original map as a view.

241. How will you sort objects by Natural order in a Java List?

We can use the Collections.sort method to sort the elements of a List in natural order. To use this method, element objects must implement the compareTo() method. We can also use a Comparator to define the natural ordering for the elements of a List, and use this custom Comparator in the sort method of the Collections class.

242. How can we get a Stream from a List in Java?

From Java 8 onwards, it is very easy to get a Stream from a List. We can use the stream() method to get a stream from a list of elements.

243. Can we get a Map from a Stream in Java?

Yes, we can create a Map from the elements of a Stream. We can use the map() method to get a Map.

```
items.stream().map(item -> item.toLowerCase())
```

In this example, we are creating a map with each item object mapped to its lowercase equivalent. This is also used in Map-Reduce implementations on a Stream.

244. What are the popular implementations of Deque in Java?

- **ArrayDeque:** It is a resizable array implementation of Deque. The capacity of ArrayDeque can increase based on the need of the program. It is not thread-safe. The iterator on ArrayDeque is fail-fast.
- **LinkedList:** This is another popular implementation of the Deque interface in Java. It is also not synchronized, so it is not thread-safe. It mainly provides the functionality of a doubly linked list.

245. What is a Thread in Java?

A thread in Java is a lightweight process that runs within another process or thread. It is an independent path of execution in an application. The JVM gives each thread its own method-call stack. When we start the JVM, Java starts one thread. This thread calls the main method of the class passed as an argument to the java call.

246. What is the priority of a Thread and how is it used in scheduling?

In Java, every Thread has a priority specified as a number between 1 to 10. The scheduler in Java schedules different threads based on their priority, known as pre-emptive scheduling. The thread with higher priority gets preference in execution over a thread with lower priority.

247. What is the default priority of a thread in Java?

In Java, a new thread gets the same priority as the parent thread that creates it. The default priority of a thread is 5 (NORM_PRIORITY).

248. What are the three different priorities that can be set on a Thread in Java?

- **MIN_PRIORITY:** The minimum priority a thread can have (value = 1).
- **NORM_PRIORITY:** The default priority assigned to a thread (value = 5).
- **MAX_PRIORITY:** The maximum priority a thread can have (value = 10).

The default priority of a thread is 5 (NORM_PRIORITY).

249. What is the purpose of join() method in Thread class?

In Java, the Thread Scheduler controls thread scheduling, but we can use the join() method on a thread to make the current thread wait for another thread to finish. When join() is used, the current thread stops executing and waits for the thread on which join() is called to finish. This ensures the current thread continues only after the joined thread has finished running. For example:

```
1 public class ThreadJoin {
2     public static void main(String[] args) throws InterruptedException {
3         Thread importantThread = new Thread(new Runnable() {
4             public void run() {
5                 // do something important
6                 System.out.println("Important thread running...");
7             }
8         });
9
10        Thread currentThread = new Thread(new Runnable() {
11            public void run() {
12                // do something else
13                System.out.println("Current thread running...");
14            }
15        });
16
17        importantThread.start(); // Line 1
18        importantThread.join(); // Line 2 - waits for importantThread to finish
19        currentThread.start(); // Line 3
20    }
21 }
```

In this example, the main thread is executing. On Line 1, a new thread called importantThread is started. At Line 2, the main thread joins importantThread and waits for it to finish before moving to Line 3. So, currentThread at Line 3 will not start until importantThread has finished.

250. What is the fundamental difference between wait() and sleep() methods?

The main difference between wait() and sleep() is that wait is an Object-level method, while sleep() is a static method in the Thread class. A waiting thread can be woken up by another thread by calling notify() on the monitor being waited on, but a sleeping thread cannot be woken up. wait() and notify() must happen within the same block that is synchronized on the monitor object. When wait() is called, the current thread releases the monitor and goes to the waiting state, and another thread calls notify() to wake it up. In the case of sleep(), the current thread does not release the monitor or locks; it just sleeps for a predefined time period.

251. Is it possible to call run() method instead of start() on a thread in Java?

Yes. We can call the run() method of a thread. But it does not work as a separate thread; it will just work as a normal object in the main thread, and there will not be context switching between threads.

252. How Multi-threading works in Java?

Java provides support for Multithreading. In a Multithreading environment, one process can execute multiple threads in parallel at the same time. In Java, you can create a process and then create multiple threads from that process. Each process can execute in parallel to perform independent tasks. Java provides methods like start(), notify(), wait(), sleep(), etc., to maintain a multithreading environment.

253. What are the advantages of Multithreading?

- Improved performance: Multithreading can improve the performance of a job.
- Simultaneous access to multiple applications: We can access multiple applications from a process using multithreading.
- Reduced number of servers required: With multithreading, fewer servers are needed, since one process can spawn multiple threads.

- Simplified coding: In certain scenarios, it is easier to code multiple threads than manage everything in a single thread.

254. What are the disadvantages of Multithreading?

- Difficult to debug: Multithreaded code is difficult to debug in case of issues.
- Difficult to manage concurrency: Multiple threads can introduce various issues.
- Difficulty of porting code: It is difficult to convert existing single-threaded code into multithreaded code.
- Deadlocks: Multithreading can lead to deadlocks when threads wait for the same resource.

255. What is a Thread in Java?

In Java, a thread is a lightweight process that runs within another process or thread. It is an independent path of execution in an application. Each thread runs in a separate stack frame. By default, Java starts one thread when the main method of a class is called.

256. What is a Thread's priority and how is it used in scheduling?

In Java, every Thread has a priority specified as an integer value. The priority value is used in scheduling to pick the thread with higher priority for execution. Threads with higher priority get more preference in execution than those with lower priority. The task scheduler schedules higher priority threads first, followed by lower priority threads.

257. What are the differences between Pre-emptive Scheduling Scheduler and Time Slicing Scheduler?

In Pre-emptive scheduling, the highest priority task keeps getting time to execute until it goes to the waiting or dead state, or a task with higher priority comes into the queue for scheduling. In Time Slicing scheduling, every task gets a predefined slice of time for execution, then it goes to the pool of tasks ready for execution. The scheduler picks up the next task for execution based on priority and other factors.

258. Is it possible to call run() method instead of start() on a thread in Java?

Yes. We can call the run() method of a thread. But it does not work as a separate thread; it will just work as a normal object in the main thread, and there will not be context-switching between threads.

259. How will you make a user thread into daemon thread if it has already started?

No. We cannot make a user thread into a daemon thread once it has already started. If we try to do so by calling setDaemon(), it will throw IllegalStateException.

260. Can we start a thread two times in Java?

No. We can call the start() method only once on a thread in Java. If we call it twice, it will throw an exception.

261. In what scenarios can we interrupt a thread?

We can interrupt a thread if we want to wake it up from the sleep or wait state.

262. In Java, is it possible to lock an object for exclusive use by a thread?

Yes. We can use a synchronized block to lock an object. The locked object is inaccessible to any other thread. Only the thread that has locked it can access it.

263. How notify() method is different from notifyAll() method?

In Java, notify() is used to unblock a specific thread that is in the waiting state, while notifyAll() is used to unblock all threads that are in the waiting state.

264. What is a daemon thread in Java?

A daemon thread in Java is a low-priority thread that does not prevent the JVM from exiting when the program finishes. The thread keeps running. Garbage Collection is an example of a daemon thread.

265. How can we make a regular thread Daemon thread in Java?

We can call the setDaemon(boolean) method to change a thread to a daemon thread before the thread starts.

266. How will you make a user thread into daemon thread if it has already started?

No. We cannot make a user thread into a daemon thread once it has already started. If we try to do so by calling `setDaemon()`, it will throw `IllegalThreadStateException`.

267. Can we start a thread two times in Java?

No. We can call the `start()` method only once on a thread in Java. If we call it twice, it will throw an exception.

268. What is a Shutdown hook in Java?

The shutdown hook is a thread that is invoked implicitly by the JVM just before shutdown. It can be used to clean up unused resources, etc. We can use `java.lang.Runtime.addShutdownHook(Thread hook)` to register a new virtual-machine shutdown hook.

269. What is synchronization in Java?

Synchronization in Java is used in multi-threaded programming to control the access of multiple threads to a shared resource. It is used to prevent deadlock between multiple threads.

270. What is the purpose of Synchronized block in Java?

A synchronized block is used in Java multi-threading environments to prevent thread interference and to avoid memory inconsistency issues. Generally, the scope of a synchronized block is smaller than the scope of a synchronized method.

271. What is static synchronization?

We can make a static method synchronized in Java by adding the `synchronized` keyword. In static synchronization, the lock is on the class, not on the object.

272. What is a Deadlock situation?

A Deadlock is a situation in which two or more threads are waiting on each other to release a resource. Each thread is waiting for a resource that is held by another waiting thread. Sometimes, there is a circular wait when more than two threads are waiting on each other's resources.

273. What is the meaning of concurrency?

Concurrency is the ability of a program to execute several programs simultaneously. This is achieved by distributing computations over multiple CPU cores of a machine or even over different machines within the same network. It can increase the speed of execution of the overall program in a multi-processor or multi-core system.

274. What is the main difference between process and thread?

Both process and thread are independent sequences of execution. The main difference is that a thread runs in a shared memory space, while a process runs in its own memory space. A process runs execution in an environment provided by the operating system and has its own set of private resources (e.g., memory, open files). A thread lives within a process and shares resources like memory and open files with other threads of the same process. This ability to share resources makes threads more suitable for tasks where performance is a significant factor.

275. What is a process and thread in the context of Java?

In Java, a process refers to the running of the Java Virtual Machine (JVM). A thread lives within the JVM and can be created or stopped by the Java application at runtime.

276. What is a Scheduler?

A scheduler is a program that implements a scheduling algorithm to manage access of processes and threads to limited resources like the CPU or an I/O channel. The goal of most scheduling algorithms is to provide load balancing for the available processes/threads and to guarantee each process/thread gets a reasonable time frame to access the requested resource exclusively.

277. What is the minimum number of Threads in a Java program?

In a JVM, each Java program is executed within the main process that starts with `java.exe`. Therefore, each Java application has at least one thread.

278. What are the properties of a Java thread?

- Identifier: A unique identifier of type long within the JVM
- Name: A name of type String
- Priority: Priority of type int
- State: A state of type java.lang.Thread.State
- Group: A thread group the thread belongs to

279. What are the different states of a Thread in Java?

- New: The thread has not yet started.
- Runnable: A thread executing in the JVM.
- Blocked: A thread waiting for a monitor lock.
- Waiting: A thread waiting indefinitely for another thread to perform a particular action.
- Timed_waiting: A thread waiting for another thread to perform an action for up to a specified waiting time.
- Terminated: A thread that has exited.

280. How will you set the priority of a thread in Java?

The priority of a thread in Java can be set by using `setPriority(int priority)` method. We can use the constant `Thread.MAX_PRIORITY` to set the maximum priority, `Thread.MIN_PRIORITY` for the minimum, and `Thread.NORM_PRIORITY` for the default priority.

281. What is the purpose of Thread Groups in Java?

In Java, every thread belongs to a group of threads. The JDK class `java.lang.ThreadGroup` provides methods to handle a whole group of Threads. With the help of these methods, we can interrupt all threads of a group or set the maximum priority of all threads of a group. So, a thread group is used for taking collective actions on a group of threads.

282. Why we should not stop a thread by calling its `stop()` method?

The `stop()` method in Thread class is a deprecated method. Its use is not recommended. When we call the `stop()` method, the thread unlocks all monitors that it has acquired. If any locked object was in an inconsistent state, this state becomes visible to all other threads. It can cause unexpected behavior when other threads work on this inconsistent object. So, calling the `stop()` method to stop a thread is not advisable.

283. How will you create a Thread in Java?

There are two main ways to create a thread in Java:

- Extend Thread class: We can extend `java.lang.Thread` class and implement the `run()` method. On calling `start()`, it will start a new thread.
- Implement Runnable interface: We can implement `java.lang.Runnable` interface and pass the implemented object to the constructor of `java.lang.Thread` class. On calling `start()`, it will start a new thread.

284. How can we stop a thread in the middle of execution in Java?

We can use a volatile variable as an indicator to stop the thread. We can create a volatile reference pointing to the current thread. This reference can be set to null by other threads to flag that the current thread should stop execution.

Sample code is as follows:

```

1 public static class MyThread extends Thread {
2     private volatile Thread threadStopper;
3
4     @Override
5     public void start() {
6         threadStopper = new Thread(this);
7         threadStopper.start();
8     }
9
10    public void stopThread() {
11        threadStopper = null;
12    }
13
14    @Override
15    public void run() {
16        Thread currThread = Thread.currentThread();
17        while (currThread == threadStopper) {
18            try {
19                Thread.sleep(100);
20            } catch (InterruptedException e) {
21                // Handle interruption if needed
22            }
23        }
24    }
25 }

```

285. How do you access the current thread in a Java program?

We can access the current thread in Java by calling the static method `currentThread()` of `java.lang.Thread` class.

Sample code is as follows:

```

1 public class MyThread {
2     public static void main(String[] args) {
3         // Get ID of Current Thread
4         long id = Thread.currentThread().getId();
5
6         // Get Name of Current Thread
7         String name = Thread.currentThread().getName();
8
9         System.out.println("Current Thread ID: " + id);
10        System.out.println("Current Thread Name: " + name);
11    }
12 }

```

286. What is Busy waiting in Multithreading?

Busy waiting is also known as busy-looping or spinning. It is a multi-threading technique in which a process repeatedly checks if a condition is true. For example, a process can keep checking if any keyboard input is available. In general, busy waiting is considered an anti-pattern that wastes processor time, so it should be avoided.

```

1 Thread thread = new Thread(new Runnable() {
2     @Override
3     public void run() {
4         long timeToStop = System.currentTimeMillis() + 1000; // Stop after 1 second
5         long currentTime = System.currentTimeMillis();
6
7         // Busy waiting
8         while (timeToStop > currentTime) {
9             currentTime = System.currentTimeMillis();
10        }
11    }
12 });

```


287. How can we prevent busy waiting in Java?

There is a simple way to prevent busy-waiting in Java. We can just put the current thread to sleep for a given amount of time. It can be done by calling the sleep() method of java.lang.Thread class. We can pass the number of milliseconds to the sleep() method as an argument.

288. Can we use Thread.sleep() method for real-time processing in Java?

Java does not guarantee that Thread.sleep() will cause the thread to sleep for exactly N number of milliseconds. Sometimes the thread can sleep for more than N milliseconds. In real-time processing, we need a precise time period for which a thread should run or sleep. Therefore, the invocation of Thread.sleep() method is not recommended for use in real-time processing.

289. Can we wake up a thread that has been put to sleep by using Thread.sleep() method?

We can use the interrupt() method of java.lang.Thread class to interrupt a thread that is in the sleep state. It will get an InterruptedException to wake up from the sleep.

Sample code is as follows:

```
1 public class ThreadInterrupt implements Runnable {
2     @Override
3     public void run() {
4         try {
5             Thread.sleep(Long.MAX_VALUE); // Simulate long-running task
6         } catch (InterruptedException e) {
7             System.out.println("Interrupted by exception!");
8         }
9     }
10
11     public static void main(String[] args) throws InterruptedException {
12         Thread myThread = new Thread(new ThreadInterrupt(), "myThread");
13         myThread.start();
14
15         System.out.println("Sleeping in main thread for 10 seconds");
16         Thread.sleep(10000);
17
18         System.out.println("Interrupting myThread");
19         myThread.interrupt();
20     }
21 }
```

290. What are the two ways to check if a Thread has been interrupted?

These are the two ways to check for thread interruption:

- In Java, a Thread can call Thread.interrupted() method to check if it has been interrupted or not.
- The other option is to call isInterrupted() method of Thread class to check if it has been interrupted or not.

291. How can we make sure that Parent thread waits for termination of Child thread?

We can use the join() method for this purpose. On calling join(), the current thread waits for the child thread to which it joins to finish.

```
1 Thread myThread = new Thread(new Runnable() {
2     @Override
3     public void run() {
4         // Task for the thread
5     }
6 });
7
8 myThread.start();
9
10 // Join on myThread (waits until myThread finishes)
11 myThread.join();
```

292. How will you handle InterruptedException in Java?

In Java, we can get InterruptedException from sleep() or join() methods. Throwing InterruptedException is a way to inform that another thread has interrupted this thread. In general, the purpose of Interrupt is to ask the current thread to stop its current execution and finish unexpectedly. Therefore, ignoring this exception by catching it and only logging it to the console or some log file is not the recommended approach. The run() method of the Runnable interface does not allow throwing any exceptions. So we cannot re-throw InterruptedException. Therefore, the correct way to handle this exception is that the run() method should check and handle this exception by itself and take appropriate action.

293. Which intrinsic lock is acquired by a synchronized method in Java?

When we mark a method as synchronized and then call this method, the method will first acquire the intrinsic lock of the object in which that method is mentioned. Once the synchronized method returns, it releases the lock. In case the synchronized method throws an exception, the intrinsic lock will be released.

```
1 public void myMethod() {  
2     synchronized (this) {  
3         // Critical section  
4     }  
5 }
```

294. Can we mark a constructor as synchronized in Java?

No. We cannot mark a constructor as synchronized. This will lead to a compiler error. The reasoning behind this is that, in this case, only the constructing thread would have access to the object being constructed.

295. Can we use primitive values for intrinsic locks?

No. Java does not allow primitive values to be used for intrinsic locks.

296. Do we have re-entrant property in intrinsic locks?

Yes. An intrinsic lock can be accessed by the same thread multiple times. So an intrinsic lock is re-entrant. If it is not allowed, then the code that acquires a lock would have to avoid acquiring the lock that it has already acquired.

297. What is an atomic operation?

An atomic operation is an operation that completes in a single step relative to other threads. An atomic operation is either executed completely or not at all. There is no halfway mark in an atomic operation.

298. Can we consider the statement i++ as an atomic operation in Java?

No. The statement i++ is not an atomic operation. It has more than one operation. First, the JVM loads the current value of i in memory. Then it increments it. Finally, it stores the new value back into variable i. The current thread that executes this operation may be interrupted between any of the above-mentioned three steps. Therefore, it is not an atomic operation.

299. What are the Atomic operations in Java?

Java language provides some basic atomic operations. These operations can be used to make sure that concurrent threads always see the same value.

- Read operations on reference variables and primitive variables (except long and double)
- Write operations on reference variables and primitive variables (except long and double)
- Read operations on all variables declared as volatile
- Write operations on all variables declared as volatile

300. Can you check if following code is thread-safe?

```
1 public class SingletonDoubleCheck {
2     private static volatile SingletonDoubleCheck instance = null;
3
4     public static SingletonDoubleCheck getInstance() {
5         if (instance == null) {
6             synchronized (SingletonDoubleCheck.class) {
7                 if (instance == null) {
8                     instance = new SingletonDoubleCheck();
9                 }
10            }
11        }
12        return instance;
13    }
14 }
```

The above-mentioned code is for creating a Singleton class. But this code is not thread-safe. In this, we check the value of instance a second time in the synchronized block. But the JIT compiler can rearrange the bytecode in such a way that the reference to SingletonDoubleCheck instance will be set before the execution of the constructor. Due to this, the method getInstance() will return an object that may not have been initialized properly. We can use the keyword volatile for instance to make this thread-safe code. Any variable that is marked as volatile will be visible to other threads only after the completion of the constructor of the object.

301. What are the minimum requirements for a Deadlock situation in a program?

For a deadlock to occur, the following are the minimum requirements:

- Mutual exclusion: There has to be a resource that can be accessed by only one thread at any point of time.
- Resource holding: One thread locks one resource and holds it, and at the same time it tries to acquire lock on another mutually exclusive resource.
- No preemption: There is no pre-emption mechanism by which resource held by a thread can be freed after a specific period of time.
- Circular wait: There can be a scenario in which two or more threads lock one resource each and they wait for each other's resource to get free. This causes circular wait among threads for the same set of resources.

302. How can we prevent a Deadlock?

To prevent a deadlock from occurring, at least one requirement for a deadlock has to be removed:

- Mutual exclusion: We can use optimistic locking to prevent mutual exclusion among resources.
- Resource holding: A thread has to release all its exclusive locks if it does not succeed in acquiring all exclusive locks for resources required.
- No preemption: We can use a timeout period for an exclusive lock to get free after a given amount of time.
- Circular wait: We can check and ensure that circular wait does not occur when all exclusive locks have been acquired by all the threads in the same sequence.

303. How can we detect a Deadlock situation?

We can use ThreadMXBean.findDeadlockedThreads() method to detect deadlocks in a Java program. This bean comes with JDK:

```

1 public class DeadlockDetector {
2     public static void main(String[] args) {
3         ThreadMXBean bean = ManagementFactory.getThreadMXBean();
4
5         // Returns null if no deadlock is found
6         long[] threadIds = bean.findDeadlockedThreads();
7
8         if (threadIds != null) {
9             // Optionally, request full stack traces (Integer.MAX_VALUE for depth)
10            ThreadInfo[] infos = bean.getThreadInfo(threadIds, Integer.MAX_VALUE);
11
12            for (ThreadInfo info : infos) {
13                StackTraceElement[] stack = info.getStackTrace();
14
15                // Log or store stack trace information
16                System.out.println("Deadlocked Thread: " + info.getThreadName() +
17                                " (ID: " + info.getThreadId() + ")");
18                for (StackTraceElement element : stack) {
19                    System.out.println("\tat " + element);
20                }
21                System.out.println();
22            }
23        } else {
24            System.out.println("No deadlocked threads detected.");
25        }
26    }
27 }

```

304. What is a Livelock?

Livelock is a scenario in which two or more threads block each other by responding to an action caused by another thread. In a deadlock situation, two or more threads wait in one specific state. In a livelock scenario, two or more threads change their state in such a way that it prevents progress on their regular work. For example, consider a scenario in which two threads try to acquire two locks. They release a lock that they have acquired when they cannot acquire the second lock. In a livelock situation, both threads concurrently try to acquire the locks. Only one thread would succeed, and the second thread may succeed in acquiring the second lock. Now both threads hold two different locks, and both threads want to have both locks. So they release their lock and try again from the beginning. This situation keeps repeating multiple times.

305. What is Thread starvation?

In a priority-based scheduling, threads with lower priority get less time for execution than higher priority threads. If a lower priority thread performs a long-running computation, it may happen that this thread does not get enough time to finish its computations just in time. In such a scenario, the thread with lower priority would starve. It will remain away from the threads with higher priority.

306. How can a synchronized block cause Thread starvation in Java?

It is not defined for synchronization which thread will enter a synchronized block. It may happen that if many threads are waiting for entry to a synchronized block, some threads may have to wait longer than other threads. Hence, these threads with lower priority will not get enough time to finish their work in time.

307. What is a Race condition?

A race condition is an unwanted situation in which a program attempts to perform two or more operations at the same time, but because of the logic of the program, the operations have to be performed in proper sequence to run the program correctly. Since it is an undesirable behavior, it is considered a bug in code. Most of the time, race condition occurs in a “check then act” scenario. Both threads check and act on the same value, but one of the threads acts in between check and act. See this example to understand race condition:

```

1  if (x == 3) { // Check
2      y = x * 5; // Act
3      // If another thread changes x between "if (x == 3)" and "y = x * 5",
4      // then y will not be equal to 15.
5  }
6

```

308. What is a Fair lock in multithreading?

In Java, there is a class `ReentrantLock` that is used for implementing fair lock. This class accepts an optional parameter `fairness`. When `fairness` is set to `true`, the `ReentrantLock` will give access to the longest waiting thread. The most popular use of fair lock is in avoiding thread starvation. Since longest waiting threads are always given priority in case of contention, no thread can starve. The downside of fair lock is the low throughput of the program. Since low priority or slow threads are getting locks multiple times, it leads to slower execution of a program. The only exception to a fair lock is the `tryLock()` method of `ReentrantLock`. This method does not honor the value of `fairness` parameter.

309. Which two methods of Object class can be used to implement a Producer Consumer scenario?

In a Producer Consumer scenario, one thread is a Producer and another thread is a Consumer. For this scenario to start working, a Consumer has to know when the Producer has produced. In `Object` class, there is a `wait()` method. A Consumer calls `wait()` to wait on Producer. The Producer uses `notify()` method of `Object` class to inform Consumer that it has produced. In this way, the processor time between produce and consume operations is freed due to the use of `wait()` and `notify()` methods.

310. How JVM determines which thread should wake up on notify()?

If multiple threads are waiting on an object's monitor, JVM awakens one of them. As per Java specification, the choice of this thread is arbitrary and it is at the discretion of the implementation. So, there is no guarantee or rule that a specific thread will be awakened by JVM on `notify()` method call.

311. Check if following code is thread-safe for retrieving an integer value from a Queue?

Sample code:

```
1 public class QueueCheck {
2     Queue<Integer> queue;
3
4     public Integer getNextInt() {
5         Integer retVal = null;
6
7         synchronized (queue) {
8             try {
9                 while (queue.isEmpty()) {
10                     queue.wait();
11                 }
12             } catch (InterruptedException e) {
13                 e.printStackTrace();
14             }
15         }
16
17         synchronized (queue) {
18             retVal = queue.poll();
19             if (retVal == null) {
20                 System.err.println("retVal is null");
21                 throw new IllegalStateException();
22             }
23         }
24
25         return retVal;
26     }
27 }
```

In the above code, `Queue` is used as object monitor to handle concurrency issues. But it may not behave correctly in a multi-threading scenario. There are two separate `synchronized` blocks in the above code. In case two threads are woken up simultaneously by another thread, both threads will enter one after another in the second `synchronized` block. Only one of the two threads will get new value from the queue and make it empty. The second thread will poll on an empty queue and it will not get any non-null return value.

312. How can we check if a thread has a monitor lock on a given object?

In Java, Thread class has a static method `holdsLock(Object objToCheck)` to check whether the thread has a lock on `objToLock` object. This method will return true if the current thread holds the lock on the `objToLock` object that was passed as an argument to this method.

313. What is the use of `yield()` method in Thread class?

The `yield()` method of Thread class is used to give a hint to the scheduler that the current thread wants to free the processor. The scheduler can either use this hint or just ignore this hint. Since the scheduler behavior is not guaranteed, it may happen that the current thread again gets the processor time. It can be used for debugging or testing purposes, but there is rarely any concrete use of this method.

314. What is an important point to consider while passing an object from one thread to another thread?

This is a multi-threading scenario. In a multi-threading scenario, the most important point is to check whether two threads can update the same object at the same time. If it is possible for two threads to update the same object at the same time, it can cause issues like race condition. So it is recommended to make the object immutable. This will help in avoiding any concurrency issues on this object.

315. What are the rules for creating Immutable Objects?

As per Java specification, the following are the rules for creating an immutable object:

- Do not provide "setter" methods that modify fields or objects referred to by fields.
- Make all fields final and private.
- Do not allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.
- If the instance fields include references to mutable objects, do not allow those objects to be changed.
- Do not provide methods that modify the mutable objects.
- Do not share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies and store references to the copies.
- Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

316. What is the use of ThreadLocal class?

ThreadLocal class provides thread-local variables. Each thread accesses only its own local variables and has its own copy of the variable. By using ThreadLocal, if thread X stores a variable with value x and another thread Y stores the same variable with the value y, then X gets x from its ThreadLocal instance and Y gets y from its ThreadLocal instance. Typically, ThreadLocal instances are private static fields that are associated with the state of a thread.

317. What are the scenarios suitable for using ThreadLocal class?

We can use an instance of ThreadLocal class to transport information within an application. One use case is to transport security or login information within an instance of ThreadLocal so that every method can access it. Another use case is to transport transaction information across an application, without using method-to-method communication.

318. How will you improve the performance of an application by multi-threading?

In an environment with more than one CPU, we can parallelize the computation tasks on multiple CPUs. This leads to parallel processing of a bigger task that takes less time due to multiple threads dividing the work among themselves. One example is that if we have to process 1000 data files and calculate the sum of numbers in each file. If each file takes 5 minutes, then 1000 files will take 5000 minutes for processing. But by using multi-threading we can process these files in

10 parallel threads. So each thread will take 100 files each. Since now work is happening in 10 parallel threads, the time taken will be around 500 minutes.

319. What is scalability in a Software program?

Scalability is the capability of a program to handle a growing amount of work or its potential to be enlarged in order to accommodate growth. A program is considered scalable if it is suitable to handle a large amount of input data, a large number of users, or a large number of nodes. When we say a program does not scale, it means that the program fails on increasing the size of the task.

320. How will you calculate the maximum speed up of an application by using multiple processors?

Amdahl's law gives the theoretical speedup in latency of the execution of a task at fixed workload. It gives the formula to compute the theoretical maximum speed up that can be achieved by providing multiple processors to an application.

If S is the theoretical speedup, then the formula is:

$$S(n) = 1 / (B + (1-B)/n)$$

where n is the number of processors and B is the fraction of the program that cannot be executed in parallel. When n converges toward infinity, the term (1-B)/n converges toward zero. Therefore, the formula can be reduced in this special case to 1/B. In general, the theoretical maximum speedup behaves in inverse proportion to the fraction that has to be executed serially. This means the lower this fraction is, the more theoretical speedup can be achieved.

321. What is Lock contention in multi-threading?

Lock contention is the situation when one thread is waiting for a lock/object that is being held by another thread. The waiting thread cannot use this object until the other thread releases the lock on that object. It is also known as Thread contention. Ideally, locks reduce the thread contention. Without locks, multiple threads can operate on the same object and cause undesirable behavior. If locking is implemented correctly, it reduces the occurrence of contention between multiple threads.

322. What are the techniques to reduce Lock contention?

There are following main techniques to reduce Lock contention:

- Reduce the scope of lock.
- Reduce object pooling.
- Reduce the number of times a certain lock can be acquired.
- Avoid synchronization at unnecessary places.
- Implement hardware supported Optimistic locking in place of synchronization.

323. What technique can be used in following code to reduce Lock contention?

The code uses Random() to get a random string and it also uses Integer to convert 42 into an object. Since these lines of code are specific to this thread, they can be moved out of the synchronization block.

```
1 Random r = new Random();
2 Integer value = Integer.valueOf(42);
3 String key = r.nextString(5); // Assuming a custom method to generate a random string
4
5 synchronized (map) {
6     map.put(key, value);
7 }
8
```

324. What is Lock splitting technique?

Lock splitting is a technique to reduce Lock contention in multi-threading. It is applicable in scenarios when one lock is used to synchronize access to different aspects of the same application. Sometimes we put one lock to protect the whole array. There can be multiple threads trying to get the lock for the same array. This single lock on the array can cause Lock

contention among threads. To resolve this, we can give one lock to each element of the array. Or we can use a modulus function to assign different locks to a small group of array elements. In this way, we can reduce the chance of Lock contention. This is Lock splitting technique.

325. Which technique is used in ReadWriteLock class for reducing Lock contention?

ReadWriteLock uses two locks: one lock for read-only operations, another lock for write operations. Its implementation is based on the premise that concurrent threads do not need a lock when they want to read a value while no other thread is trying to write. In this implementation, a read-only lock can be obtained by multiple threads. The implementation guarantees that all read operations will see only the latest updated value as soon as the write lock is released.

326. What is Lock striping?

In Lock splitting, we use different locks for different parts of the application. In Lock striping, we use multiple locks to protect different parts of the same data structure. ConcurrentHashMap class of Java internally uses different buckets to store its values. Each bucket is chosen based on the value of the key. ConcurrentHashMap uses different locks to guard different buckets. When one thread tries to access a hash bucket, it can acquire the lock for that bucket, while another thread can simultaneously acquire the lock for another bucket and access it. In a synchronized version of HashMap, the whole map has one lock. Lock striping technique gives better performance than synchronizing the whole data structure.

327. What is a CAS operation?

CAS is also known as Compare-And-Swap operation. In a CAS operation, the processor provides a separate instruction that can update the value of a register only if the provided value is equal to the current value. CAS operation can be used as an alternative to synchronization. Let's say thread T1 can update a value by passing its current value and the new value to be updated to the CAS operation. In case another thread T2 has updated the current value of the previous thread, the previous thread T1's current value is not equal to the current value of T2. Hence the update operation fails. In this case, thread T1 will read the current value again and try to update it. This is an example of optimistic locking.

328. Which Java classes use CAS operation?

Java classes like AtomicInteger or AtomicBoolean internally use CAS operations to support multi-threading. These classes are in package java.util.concurrent.atomic.

329. Is it always possible to improve performance by object pooling in a multi-threading application?

By using object pools in an application, we limit the number of new objects to be created for a class. In a single-thread operation, it can improve the performance by reusing an already created object from a pool. In a multi-threading application, an object pool has to provide synchronized access to multiple threads. Due to this, only one thread can access the pool at a time. Also, there is additional cost due to Lock contention on the pool. These additional costs can outweigh the cost saved by reuse of an object from the pool. Therefore, using an object pool may not always improve the performance in a multi-threading application.

330. How can techniques used for performance improvement in a single thread application may degrade the performance in a multi-threading application?

In a single thread application, we can use object pool for performance optimization. Whereas in a multi-threading environment, it may not be a good idea to use an object pool. Increased overhead of synchronization and lock contention can degrade the performance gained by using object pool in a multi-threading application. Another example is the implementation in which a List keeps a separate variable to hold the number of elements. This technique is useful in a single thread application where the size() method can return the value from this variable, without the need to count all the elements of the list. But in a multi-threading application, this separate variable can rather degrade the performance. This variable has to be access controlled by a lock since multiple concurrent threads can insert an element in a list. The additional cost of lock on this variable can outweigh the benefit gained by it in a multi-threading application.

331. What is the relation between Executor and ExecutorService interface?

Executor interface has only execute(Runnable) method. The implementing class of this interface has to execute the given Runnable instance passed to execute() method at some time in the future. ExecutorService interface extends Executor interface. It provides additional methods like invokeAny(), invokeAll(), shutdown(), awaitTermination(). These methods provide the ability to shut down the thread so that further requests can be rejected. Also, it provides the ability to invoke a collection of Callable tasks.

332. What will happen on calling submit() method of an ExecutorService instance whose queue is already full?

The implementation of ExecutorService will throw RejectedExecutionException when its queue is already full and a new task is submitted by calling submit() method.

333. What is a ScheduledExecutorService?

ScheduledExecutorService interface extends the interface ExecutorService. It provides various schedule() methods that can be used to submit new tasks to be executed at a given point of time. One of the schedule() methods provides the ability to schedule a one-shot task that can be executed after a given delay. Another version of schedule() method provides the ability to execute ScheduleFuture after a given amount of delay. In addition, there are scheduleAtFixedRate() and scheduleWithFixedDelay() methods that can execute an action at a periodic interval of time.

334. How will you create a Thread pool in Java?

In Java, Executors framework provides a method newFixedThreadPool(int nThreads) that can be used to create a Thread pool with a fixed number of threads.

Sample code is as follows:

```
1  import java.util.concurrent.*;
2
3  public class Example {
4      public static void main(String[] args) throws InterruptedException, ExecutionException {
5          ExecutorService myService = Executors.newFixedThreadPool(5);
6          Future<Integer>[] futureList = new Future[5];
7
8          for (int i = 0; i < futureList.length; i++) {
9              futureList[i] = myService.submit(new MyCallable());
10         }
11
12         for (int i = 0; i < futureList.length; i++) {
13             Integer retVal = futureList[i].get();
14             System.out.println(retVal);
15         }
16
17         myService.shutdown();
18     }
19
20     // Simple Callable that returns an Integer
21     static class MyCallable implements Callable<Integer> {
22         @Override
23         public Integer call() throws Exception {
24             // Do some work and return a value
25             return ThreadLocalRandom.current().nextInt(1, 101);
26         }
27     }
```

335. What is the main difference between Runnable and Callable interface?

Runnable interface defines run() method that does not return any value. Callable interface allows call() method to return a value to its caller. A Callable interface can also throw an exception in case of an error. Also, Callable is a newer addition to Java since version 1.5.

336. What are the uses of Future interface in Java?

We can use Future interface to represent the result of an asynchronous computation. These are the operations whose result is not immediately available. Therefore, Future interface provides isDone() method to check if the asynchronous

computation has finished or not. We can also check if the task was cancelled by calling `isCancelled()` method. Future also provides `cancel()` method to attempt the cancellation of a task.

337. What is the difference in concurrency in HashMap and in Hashtable?

In a Hashtable class, all methods are synchronized. In a HashMap implementation, all the methods are not synchronized. Therefore, Hashtable is a thread-safe collection. HashMap is not a thread-safe collection. In a multi-threading environment, it is not advisable to use regular HashMap. We can use ConcurrentHashMap class in multi-threading applications.

338. How will you create synchronized instance of List or Map Collection?

In Java, Collections class provides methods to synchronize any collection. It also provides `synchronizedList(List)` and `synchronizedMap(Map)` methods that can be used to convert a List or Map to a synchronized instance.

339. What is a Semaphore in Java?

Semaphore class in Java is used to implement a counting semaphore. It is used to restrict the number of threads that can access a physical or logical resource. A Semaphore maintains a set of permits that should be acquired by competing threads. We can also use it to control how many threads can access the critical section of a program or a resource concurrently. The first argument in Semaphore constructor is the total number of permits available. Each invocation of `acquire()` method tries to obtain one of the available permits. The `acquire()` method is used to acquire a permit from the semaphore. If we pass number of permits required to `acquire()` method, then it blocks the thread until that number of permits are available. Once a thread has finished its work, we can use `release()` method to release the permits.

340. What is a CountdownLatch in Java?

CountDownLatch class helps in implementing synchronization in Java. It is used to implement the scenarios in which one or more threads have to wait until other threads have reached the same state such that all threads can start. There is a synchronized counter that is decremented until it reaches the value zero. Once it reaches zero, it means that all waiting threads can proceed now. It is a versatile tool that can be used for other synchronization scenarios as well. It can also work as on/off latch or gate. All threads invoking `await()` method wait at the gate until it is opened by a thread invoking `countdown()` method.

341. What is the difference between CountdownLatch and CyclicBarrier?

CyclicBarrier takes an optional Runnable task that is run once the common barrier condition is achieved. CountdownLatch is used in simple use cases where a simple start-stop is required. A CyclicBarrier is useful in complex scenarios where more coordination is required. E.g. MapReduce algorithm implementation. CyclicBarrier resets the internal value to the initial value once the value reaches zero. CyclicBarrier can be used to implement the scenarios in which threads have to wait for each other multiple times.

342. What are the scenarios suitable for using Fork/Join framework?

ForkJoinPool class is in the center of Fork/Join framework. It is a thread pool that can execute instances of ForkJoinTask. ForkJoinTask class provides the `fork()` and `join()` methods. The `fork()` method is used to start the asynchronous execution of a task. The `join()` method is used to await the result of the computation. Therefore, divide-and-conquer algorithms can be easily implemented with Fork/Join framework.

343. What is the difference between RecursiveTask and RecursiveAction class?

RecursiveAction class has `compute()` method that does not have to return a value. RecursiveAction can be used when the action has to directly operate on a data structure. It does not need to return any computed value. In RecursiveTask class, `compute()` method always returns a value. Both RecursiveTask and RecursiveAction classes are used in ForkJoinTask implementations.

344. In Java 8, can we process stream operations with a Thread pool?

In Java 8, Collections provide `parallelStream()` method to create a stream that can be processed by a Thread pool. We can also call the intermediate method `parallel()` on a given stream to convert it into a sequential stream of parallel tasks.

345. What are the scenarios to use parallel stream in Java 8?

A parallel stream in Java 8 has a much higher overhead compared to a sequential one. It takes a significant amount of time to coordinate the threads. We can use parallel stream in the following scenarios:

- When there are a large number of items to process and the processing of each item takes time and is parallelizable.
- When there is a performance problem in the sequential processing.
- When current implementation is not already running in a multi-thread environment. If there is already a multi-threading environment, adding parallel stream can degrade the performance.

346. How Stack and Heap work in Java multi-threading environment?

In Java, Stack and Heap are memory areas available to an application. Every thread has its own stack. It is used to store local variables, method parameters, and call stack. Local variables stored in the stack of one thread are not visible to another thread. Whereas, Heap is a common memory area in JVM. Heap is shared by all threads. All objects are created inside heap. To improve performance, a thread can cache the values from heap into their stack. This can create problem if the same variable is modified by more than one thread. In such a scenario, we should use the volatile keyword to mark a variable volatile. For a volatile variable, the thread always reads the value from main memory.

347. How can we take Thread dump in Java?

The steps to take Thread dump of a Java process depend on the operating system. On taking Thread dump, Java writes the state of all threads in log files or standard error console. We can press Ctrl + Break key together to take thread dump in Windows. We can execute kill -3 command for taking Thread dump on Linux. Another option to take Thread dump is jstack tool. We can pass process id of Java process to this tool for taking Thread dump. This is the simple one, -Xss parameter is used to control stack size of thread in Java. You can see this list of JVM options to learn more about this parameter.

348. Which parameter can be used to control stack size of a thread in Java?

We use -Xss parameter to control the stack size of a thread in Java. If we set it as 1 MB, then every thread will get 1MB of stack size.

349. There are two threads T1 and T2? How will you ensure that these threads run in sequence T1, T2 in Java?

In Java, there are multiple ways to execute threads in a sequence. One of the simplest ways for sequencing is join() method of Thread class. We can call join() method to start a thread when another thread has finished. We start with the last thread to execute first and make this thread join on the next thread. In this case, we start thread T2 first and then call T1.join() so that thread T2 waits for thread T1 to finish execution. Once T1 completes execution, T2 thread starts executing.

350. What are the new features released in Java 8?

The new features released in Java 8 are:

- Lambda Expression
- Stream API
- Date and Time API
- Functional Interface
- Interface Default and Static Methods
- Optional
- Base64 Encoding and Decoding
- Nashorn JavaScript Engine
- Collections API Enhancements
- Concurrency Enhancements
- Fork/Join Framework Enhancements
- Spliterator
- Internal Iteration
- Type Annotations and Repeatable Annotations

- Method Parameter Reflection
- JVM Parameter Changes

351. What are the main benefits of new features introduced in Java 8?

The main benefits of Java 8 features are:

- Support for functional programming by Lambda and Streams
- Ease of high volume data processing by Streams
- Ease of use by getting Parameter names through Reflection
- Reusable code with enhanced Collection APIs
- Smart exception handling with Optional
- Control on JVM with new Parameters
- Enhanced encryption support with Base 64
- Faster execution with Nashorn JavaScript engine support

352. What is a Lambda expression in Java 8?

Lambda expression is an anonymous function. It is like a method that does not need any access modifiers, name or return value declaration. It accepts a set of input parameters and returns a result. Lambda expression can be passed as a parameter in a method. So we can treat code in Lambda expression as data. This piece of code can be passed to other objects and methods.

353. What are the three main parts of a Lambda expression in Java?

Three main parts of a Lambda expression are:

- Parameter list: A Lambda expression can have zero or more parameters. Parameter list is optional to Lambda.
- Lambda arrow operator: “->” is known as Lambda arrow operator. It separates the list of parameters and the body of Lambda.
- Lambda expression body: The piece of code that we want to execute is written in Lambda expression body.

E.g. In following example:

```
Arrays.asList( "a", "b", "d" ).forEach( e -> System.out.println( e ) );
```

Parameter list = e

Arrow = ->

Body = System.out.println(e)

354. What is the data type of a Lambda expression?

A Lambda expression fulfills the purpose of passing code as data. The data type of a Lambda expression is a Functional interface. In most of the cases, this is java.lang Runnable interface.

355. What is the meaning of following lambda expression?

```
(e -> System.out.println( e ));
```

This Lambda expression takes a parameter e and prints it via System.out.

356. Why did Oracle release a new version of Java like Java 8?

The main theme of Java 8 is support for functional programming. With increase in database size and growth of multi-core CPU servers, there is need for Java to support such large-scale systems. With new features of Java 8, it is possible to create functional programs to interact efficiently with Big Data systems. Support for Streams is very helpful in this regard. Lambda expressions are very useful for cloud computing where we can pass code as data and run the same code on multiple servers. Optional is a best practice that is borrowed from Google Guava library for handling the exceptional cases. This has made programs more robust with support for edge cases.

357. What are the advantages of a lambda expression?

We can pass a lambda expression as an object to a method. This reduces the overhead involved in passing an anonymous class. We can also pass a method as a parameter to another method using lambda expressions.

358. What is a Functional interface in Java 8?

A Functional interface in Java is an interface that has exactly one abstract method. It can have default methods with implementation. A default method is not abstract. In Java 8, `java.lang.Runnable` and `java.util.concurrent.Callable` are two very popular Functional interfaces.

359. What is a Single Abstract Method (SAM) interface in Java 8?

A Functional interface is also known as Single Abstract Method Interface, since it has exactly one abstract method.

360. How can we define a Functional interface in Java 8?

To define a Functional interface in Java 8, we can create an interface with exactly one abstract method. Another way is to mark an interface with annotation `@FunctionalInterface`. Even with the annotation, we have to follow the rule of exactly one abstract method. The only exception to this rule is that if we override `java.lang.Object` class's method as an abstract method, then it does not count as an abstract method.

361. Why do we need Functional interface in Java?

Functional Interfaces are mainly used in Lambda expressions, Method reference and constructor references. In functional programming, code can be treated as data. For this purpose, Lambda expressions are introduced. They can be used to pass a block of code to another method or object. Functional Interface serves as a data type for Lambda expressions. Since a Functional interface contains only one abstract method, the implementation of that method becomes the code that gets passed as an argument to another method.

362. Is it mandatory to use @FunctionalInterface annotation to define a Functional interface in Java 8?

No, it is not mandatory to mark a Functional interface with `@FunctionalInterface` annotation. Java does not impose this rule. But, if we mark an interface with `@FunctionalInterface` annotation, then Java Compiler will give us an error in case we define more than one abstract method inside that interface.

363. What are the differences between Collection and Stream API in Java 8?

Main differences between Collection and Stream API in Java 8 are:

- Version: Collection API is in use since Java 1.2. Stream API is a recent addition to Java in version 8.
- Usage: Collection API is used for storing data in different kinds of data structures. Stream API is used for computation of data on a large set of objects.
- Finite: With Collection API, we can store a finite number of elements in a data structure. With Stream API, we can handle streams of data that can contain infinite number of elements.
- Eager vs. Lazy: Collection API constructs objects in an eager manner. Stream API creates objects in a lazy manner.
- Multiple consumption: Most of the Collection APIs support iteration and consumption of elements multiple times. With Stream API, we can consume or iterate elements only once.

364. What are the main uses of Stream API in Java 8?

Main uses of Stream API in Java 8 are:

- It helps in using data in a declarative way. We can make use of database functions like Max, Min etc., without running a full iteration.
- It makes good use of multi-core architectures without worrying about multi-threading code.
- We can create a pipeline of data operations with Java Stream that can run in a sequence or in parallel.
- It provides support for group by, order by etc. operations.
- It supports writing code in Functional programming style.
- It provides parallel processing of data.

365. What are the differences between Intermediate and Terminal Operations in Java 8 Streams?

Main differences between Intermediate and Terminal Stream operations are as follows:

- Evaluation: Intermediate operations are not evaluated until we chain it with a Terminal Operation of Stream. Terminal Operations can be independently evaluated.
- Output: The output of Intermediate Operations is another Stream. The output of Terminal Operations is not a Stream.
- Lazy: Intermediate Operations are evaluated in lazy manner. Terminal Operations are evaluated in eager manner.
- Chaining: We can chain multiple Intermediate Operations in a Stream. Terminal Operations cannot be chained multiple times.
- Multiple: There can be multiple Intermediate operations in a Stream operation. There can be only one Terminal operation in Stream processing statement.

366. What is a Spliterator in Java 8?

A Spliterator is a special type of Iterator to traverse and partition the elements of a source in Java. A source can be a collection, an IO channel or a generator function. A Spliterator may traverse elements individually or sequentially in bulk.

367. What are the differences between Iterator and Spliterator in Java 8?

Main differences between Iterator and Spliterator are as follows:

- Spliterator can be used with Streams in Java 8, whereas Iterator is just used with Collection.
- Spliterator uses Internal Iteration to iterate Streams. Iterator uses External Iteration to iterate Collections.
- Spliterator can iterate Streams in Parallel as well as Sequential manner. Iterator only iterates in Sequential manner.
- Spliterator can traverse elements individually as well as in bulk. Iterator only iterates elements individually.

368. What is Type Inference in Java 8?

A Java compiler can see each method's invocation and its declaration to determine what type arguments are required for invocation. By Type Inference, Java can determine the types of the arguments as well as the type of the result being returned. Type inference algorithm also tries to find the most specific type that can work with all types of arguments.

369. Does Java 7 support Type Inference?

Yes, Java 7 supports Type Inference. In Java 8, Oracle has enhanced the Type Inference concept. Now it can be used to define Lambda expressions, functions, and Method references.

370. How does Internal Iteration work in Java 8?

In an Iterator, the fundamental question is which party controls the iteration—is it Iterator or the Collection on which iterator runs. When a Collection controls the iterator, then it is called External Iteration. When the Iterator controls the iteration, then it is called Internal Iteration. In case of Internal Iteration, the client hands over an operation to Iterator and the Iterator applies the operation to all the elements in aggregate. Internal Iteration is easier to implement, since the Iterator does not have to store the state of the collection.

371. What are the main differences between Internal and External Iterator?

Main differences between Internal and External Iterator are as follows:

- An Internal Iterator controls the iteration itself. In an External Iterator, the collection controls the iteration.
- Internal Iterator can iterate elements individually as well as in bulk (like `forEach`). External Iterator iterates elements one by one.
- Internal Iterator does not have to iterate elements only sequentially. External Iterator always iterates sequentially.
- Internal Iterator supports declarative programming style that goes well with functional programming. External Iterator follows imperative style OOPS programming.
- Some people consider Internal Iterator code more readable than that of External Iterator.

372. What are the main advantages of Internal Iterator over External Iterator in Java 8?

Some of the main advantages of Internal Iterator are:

- Internal Iterator is based on functional programming, therefore it can work on declarative style code.
- There is no need to sequentially iterate elements in Internal Iterator.
- Code is more readable and concise in Internal Iterator.
- Internal Iterator supports concurrency and parallel processing.

373. What are the applications in which we should use Internal Iteration?

We need Internal Iterator in applications that require high performance, parallel processing, fast iteration, and bulk operations support. Also, in Internal Iteration applications, we do not have much control over iteration. The other features like parallel processing, etc. become more important.

374. What is the main disadvantage of Internal Iteration over External Iteration?

Internal Iteration has many advantages over External Iteration. But it has one big disadvantage. Since Java API is responsible for iterating in Internal Iterator, the developer does not get any control over iteration.

375. Can we provide implementation of a method in a Java Interface?

Before Java 8, it was not allowed to provide implementation of a method in an Interface. Java 8 has introduced the flexibility of providing implementation of a method in an interface. There are two options for that:

- Default Method: We can give default implementation of a method.
- Static Method: We can create a static method in an interface and provide implementation.

376. What is a Default Method in an Interface?

In Java 8, we can provide implementation of a method in an Interface and mark this method with the default keyword. In this way, this implementation of the method becomes the default behavior for any class implementing the interface.

377. Why do we need Default method in a Java 8 Interface?

Default methods in an Interface provide backward compatibility feature in Java 8. Let's say there is an interface Car that is implemented by BMW, Chevrolet, and Toyota classes. Now a Car needs to add capability for flying. It will require a change in Car interface. Some of the car classes that do not have flying capability may fail. Therefore, a default implementation of flying methods is added in Car interface so that cars with no flying capability can continue to implement the original Car interface.

378. What is the purpose of a Static method in an Interface in Java 8?

A Static method in an Interface is a utility or helper method. This is not an object-level instance method. Some of the uses of Static method in an Interface are:

- Single Class: There is no need to create a separate Utils class for storing utility or helper methods. We can keep these methods in the same interface.
- Encapsulation: With Static methods, the complete behavior of a Class is encapsulated in the same class. There is no need to maintain multiple classes.
- Extension: It is easier to extend a Class/API. If we extend a collection like ArrayList, we get all the methods. We need not extend Collections class also.

379. What are the core ideas behind the Date/Time API of Java 8?

There are three core ideas behind the Date/Time API of Java 8:

- Immutable-value classes: The new API avoids thread-safety and concurrency issues by ensuring that all the core classes are immutable and represent well-defined values.
- Domain-driven design: The new API is modeled on a precise domain with classes that represent different use cases for Date and Time.
- The emphasis on domain-driven design offers benefits like clarity and understandability.
- Separation of chronologies: The new API allows people to work with different calendar systems. It supports the needs of users in different areas of the world like Japan or Thailand that don't follow ISO-8601.

380. What are the advantages of new Date and Time API in Java 8 over old Date API?

Some of the advantages of Java 8 Date Time API over existing Date API are:

- **Concurrency:** Existing Date Time classes (such as `java.util.Date` and `SimpleDateFormat`) are not thread-safe. This does not work well in concurrent applications. In new Date Time API, developer does not have to deal with concurrency issues while writing date-handling code.
- **Better Design:** Date/Time classes prior to Java 8 have poor API design. For example, years in `java.util.Date` start at 1900, months start at 1, and days start at 0. It is not very intuitive. Java 8 Date Time API handles it very well.
- **No need for 3rd Party Libraries:** With the popularity of third-party Date/Time libraries like Joda Time, Java has to make its native Date/Time API comparable. Now we can use the Java API instead of using 3rd party libraries.

381. What are the main differences between legacy Date/Time API in Java and Date/Time API of Java 8?

Main differences between legacy Date/Time API and Java 8 Date/Time API are:

- Old API is not thread-safe. Java 8 API is thread-safe.
- Old API has many mutable objects. New Java 8 API is based on immutable objects.
- Performance of old API is not good. New Java 8 Date/Time API gives better performance.
- Old API is less readable and maintainable. New Java 8 API is very well designed and is more readable.
- Old API has month values from 0 to 11. New API has months from 1 to 12.

382. How can we get duration between two dates or time in Java 8?

In Java 8, we have a new class `Duration` that provides the utility of computing duration between two dates. We can call the static method `Duration.between(date1, date2)` to get the time period in hours, minutes, days, etc. between `date1` and `date2`.

383. What is the new method family introduced in Java 8 for processing of Arrays on multi core machines?

Java 8 has enhanced the `Arrays` class with methods that can run efficiently on multi core machines. These methods start with the keyword `parallel`. Eg: `Arrays.parallelSetAll()`, `Arrays.parallelSort()`, etc. This parallel set of methods provides parallel processing of Arrays that can run Java code very fast on a multi core machine.

384. How does Java 8 solve Diamond problem of Multiple Inheritance?

In Multiple Inheritance, if a class extends more than one class with two different implementations of the same method, then it causes the Diamond problem.

```
1  interface A {
2      default void display() {
3          System.out.println("Display from A");
4      }
5  }
6
7  interface B extends A { }
8
9  interface C extends A { }
10
11 public class D implements B, C {
12     @Override
13     public void display() {
14         // Resolve ambiguity by explicitly calling one interface's default method
15         B.super.display();
16     }
17
18     public static void main(String[] args) {
19         new D().display(); // Output: Display from A
20     }
21 }
```


385. What are the differences between Predicate, Supplier and Consumer in Java 8?

The subtle difference between Predicate, Supplier and Consumer in Java 8 is as follows:

- Predicate is an anonymous function that accepts one argument and returns a result.
- Supplier is an anonymous function that accepts no argument and returns a result.
- Consumer is an anonymous function that accepts one argument and returns no result.

386. Is it possible to have default method definition in an interface without marking it with default keyword?

No, we have to always mark a default method in interface with the default keyword. If we create a method with implementation in an interface, but do not mark it as default, then we will get a compile-time error.

387. Can we create a class that implements two Interfaces with default methods of same name and signature?

No, it is not allowed to create a class that implements interfaces with same name default methods. It will give us compile-time error for duplicate default methods.

388. How Java 8 supports Multiple Inheritance?

In Multiple Inheritance, a class can inherit behavior from more than one parent class. Prior to Java 8, a class can implement multiple interfaces but extend only one class. In Java 8, we can have method implementation within an interface. So an interface behaves like an Abstract class. Now if we implement more than one interface with method implementation in a class, it means we are inheriting behavior from multiple abstract classes. That is how we get Multiple Inheritance in Java 8.

389. In case we create a class that extends a base class and implements an interface. If both base class and interface have a default method with same name and arguments, then which definition will be picked by JVM?

In such a scenario, JVM will pick the definition in base class.

390. If we create same method and define it in a class, in its parent class and in an interface implemented by the class, then which definition will be invoked if we access it using the reference of Interface and the object of class?

In all the cases, method defined in the class will be invoked.

391. Can we access a static method of an interface by using reference of the interface?

No, a static method of interface has to be invoked by using the name of the interface.

392. How can you get the name of Parameter in Java by using reflection?

Java 8 has introduced a method `Parameter.getName()` to get the name of a parameter by using reflection. Before using this feature, we need to turn on this feature in Java compiler. To turn on this feature, just run `javac` with `-parameters` argument. To verify the availability of this feature, we can use `Parameter.isNamePresent()` method.

393. What is Optional in Java 8?

Optional is a container object that may have a null or non-null value. If it has a value then `isPresent()` method returns true. If a value is present, we can call `get()` method to get the value. Else we will get nothing. It is very useful in handling data that has null values.

394. What are the uses of Optional?

Some of the uses of Optional in Java are:

- We can use Optional to avoid `NullPointerException` in an application.
- Optional performs Null check at compile time, so we do not get run time exception for a null value.
- Optional reduces the codebase pollution by removing unnecessary null checks.
- Optional can also be used to handle default case for data when a value is null.

395. Which method in Optional provides the fallback mechanism in case of null value?

In case, an Optional has null value, we can use `orElseGet()` method as fallback mechanism. If we implement `orElseGet()` method, it will be invoked when the value of Optional is null.

396. How can we get current time by using Date/Time API of Java 8?

In Java 8, we can use Clock class to get the current time. Instead of using old method System.currentTimeMillis(), we can create a Clock object and call millis() method to get the current time in milliseconds. We can also call instant() method on Clock object to get the current time in a readable format.

397. Is it possible to define a static method in an Interface?

Yes, from Java 8, an Interface can also have a static method.

398. How can we analyze the dependencies in Java classes and packages?

Java 8 comes with a new command line tool jdeps that can help in analyzing the package-level and class-level dependencies. We can pass a jar file name or a class name as an argument to this tool. It will list all the dependencies of that jar or class.

399. What are the new JVM arguments introduced by Java 8?

In Java 8, PermGen space of ClassLoader is removed. It has been replaced with MetaSpace. Now we can set the initial and maximum size of MetaSpace. The JVM options -XX:PermSize and -XX:MaxPermSize are replaced by -XX:MetaSpaceSize and -XX:MaxMetaspaceSize respectively in Java 8.

400. What are the popular annotations introduced in Java 8?

Some of the popular annotations introduced in Java 8 are:

- @FunctionalInterface: This annotation is used to mark an interface as Functional Interface. As mentioned earlier, a FunctionalInterface can be used for lambda expressions.
- @Repeatable: This annotation is used for marking another annotation. It indicates that the marked annotation can be applied multiple times on a type.

401. What is a StringJoiner in Java 8?

StringJoiner is a new class in Java 8 that can be used to create a String. It can construct a sequence of characters separated by a delimiter. It can also optionally add a prefix and suffix to this sequence. We can use this sequence to get a String.

For example, the String "[One:Two:Three]" may be constructed as follows:

```
StringJoiner sj = new StringJoiner(":", "[", "]");  
sj.add("One").add("Two").add("Three");  
String desiredString = sj.toString();
```

402. What is the type of a Lambda expression in Java 8?

The type of a lambda expression depends on the context it is being used. A lambda is like a method reference. It does not have a type of its own. Generally, a Lambda is an instance of a Functional Interface.

403. What is the target type of a lambda expression?

The target type of a lambda expression represents a type to which the expression can be converted. The target type for a lambda expression is a functional interface. The lambda expression must have the same parameter type as the parameter in the function of the interface. It must also return a type compatible with the return type of the function.

404. What are the main differences between an interface with default method and an abstract class in Java 8?

- Instance variable: An interface cannot have instance variables. An abstract class can have instance variables.
- Constructor: An interface cannot have a constructor. An abstract class can have a constructor.
- Concrete Method: An interface cannot have concrete methods other than default method. An abstract class is allowed to define concrete methods with implementation.
- Lambda: An interface with exactly one default method can be used for lambda expression. An abstract class cannot be used for lambda expression.

405. Is there any difference between a = a + b and a += b expressions?

When we add two integral variables, e.g., variables of type byte, short, or int in Java, they are first promoted to int type, and then addition happens. The += operator implicitly casts the result of addition into the type of variable used to hold the result.

What happens when you put return statement or System.exit() in try or catch block? Will finally block execute?

Finally block executes even if we put a return statement in the try or catch block. But finally block does not execute if you call System.exit() from try or catch block.

406. What does the expression 1.0 / 0.0 return? Will there be any compilation error?

The answer is that 1.0 / 0.0 will compile successfully and will not throw ArithmeticException. It will just return Double.INFINITY.

407. Can we use multiple main methods in multiple classes?

Yes. When we start an application in Java, we just mention the class name to be run to java command. The JVM looks for the main method only in the class whose name is passed to the java command. Therefore, there is no conflict amongst the multiple classes having main method.

408. Does Java allow you to override a private or static method?

You cannot override a private or static method in Java. If we create a similar method with the same return type and same method arguments in a child class, then it will hide the superclass method. This is known as method hiding. Also, you cannot override a private method in a subclass because a private method is not visible even in a subclass. Therefore, what you can do is create another private method with the same name in the child class. So in both cases, it is not method overriding. It is either method hiding or a new method.

409. What happens when you put a key object in a HashMap that is already present?

In a HashMap there are buckets in which objects are stored. Key objects with the same hashCode go to the same bucket. If you put the same key again in a HashMap, then it will replace the old mapping because HashMap doesn't allow duplicate keys. The same key will have the same hashCode as the previous key object. Due to the same hashCode, it will be stored at the same position in the bucket.

410. How can you make sure that N threads can access N resources without deadlock?

By resource ordering you can prevent deadlock. If in your program you always acquire resources in a particular order and release resources in the reverse order, then you can prevent deadlock. So a thread waiting for the same resource cannot get into deadlock while another thread is trying to get it and holding the resource required by the first thread. If both of them release the resources in the right order, one of them can acquire it to finish the work.

411. How can you determine if JVM is 32-bit or 64-bit from a Java Program?

You can find JVM bit size (32-bit or 64-bit) by running the java command from the command prompt, or you can get it from a Java program. Sun has a Java System property to determine the bit size of the JVM:

```
sun.arch.data.model=32 // 32 bit JVM
```

```
sun.arch.data.model=64 // 64 bit JVM
```

You can use System.getProperty("sun.arch.data.model") to determine the JVM bit size from your Java program.

412. What is the right data type to represent Money (like Dollar/Pound) in Java?

To represent money you need decimal points in the numbers like \$1.99. BigDecimal class provides good methods to represent Money. Using BigDecimal, you can do the calculation with decimal points and correct rounding. But using BigDecimal is a little bit high on memory usage. You can also use double with predefined precision, but calculation on double can give erroneous results.

413. How can you do multiple inheritances in Java?

Java supports multiple inheritances by allowing an interface to extend other interfaces. You can implement more than one interface, but you cannot extend multiple classes. So Java doesn't support multiple inheritances of implementation. In Java 8, the default method breaks the rule of multiple inheritance behavior.

414. Is ++ operation thread-safe in Java?

No, the ++ operator is not a thread-safe operation. It involves multiple instructions like reading a value, incrementing it, and storing it back into memory. These instructions can overlap between multiple threads, causing issues in multi-threading.

415. How can you access a non-static variable from a static context?

You cannot access a non-static variable from the static context in Java. If you write code like that, you will get a compile-time error. This is a common problem for beginner Java programmers when they try to access an instance variable inside the main method in a class. Since the main method is static in Java and instance variables are non-static, you cannot access instance variables inside main. The solution is to create an instance of the object and then access the instance variables.

416. Let's say there is a method that throws NullPointerException in the superclass. Can we override it with a method that throws RuntimeException?

You can throw a superclass of RuntimeException in an overridden method, but you cannot do the same if it is a checked Exception.

417. How can you mark an array volatile in Java?

You can mark an array volatile in Java, but it makes only the reference to the array volatile, not the whole array. If one thread changes the reference variable to point to another array, it will provide a volatile guarantee. But if multiple threads are changing individual array elements, they won't have the same reference due to the reference itself being volatile.

418. What is a thread local variable in Java?

A thread-local variable is a variable restricted to a specific thread. It is like a thread's own copy of a variable that is not shared among multiple threads. Java provides the ThreadLocal class to support thread-local variables. To achieve thread-safety, you can use it. To avoid any memory leak, it is always good to remove a thread-local variable once its work is done.

419. What is the difference between sleep() and wait() methods in Java?

In Java, we use these methods to pause the currently running thread. The difference is simple: sleep() is meant for a short pause because it doesn't release the lock. wait() is meant for conditional wait and it can release a lock that can be acquired by another thread to change the condition on which it is waiting.

420. Can you create an Immutable object that contains a mutable object?

In Java, it is possible to create an Immutable object that contains a mutable object. We should not share the reference of the mutable object since it is inside an immutable object. Instead, we can return a copy of it to other methods.

421. How can you convert an Array of bytes to String?

You can convert an Array of bytes to a String object by using the String constructor that accepts byte[]. You need to make sure that the right character encoding is used, otherwise you may get different results after conversion.

422. What is the difference between CyclicBarrier and CountdownLatch class?

CyclicBarrier and CountdownLatch classes were introduced in Java 5. You can reuse CyclicBarrier even if it is broken, but you cannot reuse CountdownLatch in Java.

423. What is the difference between StringBuffer and StringBuilder?

StringBuilder was introduced in Java 5. The main difference between both of them is that StringBuffer methods (e.g., length(), capacity(), append()) are synchronized, but corresponding methods in StringBuilder are not synchronized. Due to this difference, concatenation of String using StringBuilder is faster than StringBuffer. Now it is considered bad practice to use StringBuffer because, in most scenarios, we perform string concatenation in the same thread.

424. Which class contains clone method? Cloneable or Object class?

The clone() method is defined in Object class. Cloneable is a marker interface that doesn't contain any method.

425. How will you take thread dump in Java?

- Linux/Unix: Use `kill -3 PID`, where PID is the process id of the Java process. It will give the thread dump of the Java process.
- Windows: Press `Ctrl + Break`. This will instruct JVM to print thread dump in standard out or err. It can go to console or log file depending upon your application configuration.

426. Can you cast an int variable into a byte variable? What happens if the value of int is larger than byte?

An int is 32-bit in Java, but a byte is just 8-bit. You can cast an int to byte, but you will lose higher 24 bits of int while casting because a byte can hold only the first 8 bits of int. The remaining 24 bits ($32-8=24$) will be lost.

427. In Java, can we store a double value in a long variable without explicit casting?

No, you cannot store a double value into a long variable without casting it to long. The range of double is more than that of long, so you need to type cast.

428. What will this return: `5*0.1 == 0.5`? true or false?

The answer is false because floating point numbers cannot be represented exactly in Java, so $5*0.1$ is not the same as 0.5.

429. Out of an int and Integer, which one takes more memory?

An Integer object takes more memory than an int in Java. An Integer is an object and it stores meta-data overhead about the object. An int is a primitive type so it takes less memory and there is no meta-data overhead.

430. Can we use String in the switch case statement in Java?

Yes. From Java 7 onwards, String can be used in switch case statements. This gives convenience to programmers. Internally, the hash code of String is used for the switch statement.

431. Can we use multiple main methods in the same class?

Yes. You can have multiple methods with the name main in the same class. But there should be only one main method with the signature `public static void main(String[] args)`. JVM looks for main with this signature only. Other methods with the name main in the same class are just ignored.

432. When creating an abstract class, is it a good idea to call abstract methods inside its constructor?

No, you should avoid calling abstract methods in the constructor of an abstract class. It can restrict how these abstract methods can be implemented by child classes. Many IDEs give an “Overridable method call in constructor” warning for such implementation. This is a problem of object initialization order. The superclass constructor will run before the child class constructor. It means child class is not yet initialized, but due to the presence of an overridden method in the superclass, the overridden method of the subclass is called when the subclass is not fully initialized.

433. How can you do constructor chaining in Java?

When you call one constructor from another constructor of the same class, it is known as constructor chaining in Java. When you have multiple overloaded constructors in a class, you can do constructor chaining.

434. How can we find the memory usage of JVM from Java code?

You can use memory management related methods provided in `java.lang.Runtime` class to get the free memory, total memory, and maximum heap memory in Java. By using these methods, you can find out how much of the heap is used and how much heap space still remains.

- `Runtime.freeMemory()` returns amount of free memory in bytes.
- `Runtime.totalMemory()` returns total memory in bytes.
- `Runtime.maxMemory()` returns maximum memory in bytes.

435. What is the difference between `x == y` and `x.equals(y)` expressions in Java?

The `x == y` expression does object reference matching if both are objects and only returns true if both are pointing to the same object in the heap space. The `x.equals(y)` expression is used for logical mapping and it is expected from an object to

override this method to provide logical equality. For example, a Book object may be logically equal to another copy of the same Book, but it is a different object, which will be false while doing `x == y`.

436. How can you guarantee that the garbage collection takes place?

No, you cannot guarantee the garbage collection in Java. Java documentation explicitly says that `GarbageCollection` is not guaranteed. You can call `System.gc()` to request garbage collection; however, that's what it is—a request. It is up to the GC's discretion to run.

437. What is the relation between `x.hashCode()` method and `x.equals(y)` method of Object class?

`x.hashCode()` method returns an int hash value corresponding to an object instance. It is used in hashCode-based collection classes like `Hashtable`, `HashMap`, `LinkedHashMap`, etc. `hashCode()` method is also related to `equals()` method. As per Java specification, two objects which are equal to each other using `equals()` method must have the same hash code. Therefore, two objects with the same hashCode may or may not be equal to each other. But two equal objects should have the same hash code.

438. What is a compile time constant in Java?

A compile time constant is a public static final variable. The public modifier is optional here. At compile time, they are replaced with actual values because the compiler knows their value up-front and also knows that it cannot be changed during run time. So they are constants.

439. Explain the difference between fail-fast and fail-safe iterators?

The main difference between fail-fast and fail-safe iterators is whether or not the collection can be modified while it is being iterated.

- Fail-safe iterators allow modification of the collection in an iteration task. Fail-fast iterators do not allow any modification to the collection during iteration.
- During iteration, fail-fast iterators fail as soon as they realize that the collection has been modified. Modification can be addition, removal, or update of a member. It will throw a `ConcurrentModificationException`. Examples: `ArrayList`, `HashSet`, and `HashMap` are fail-fast.
- Fail-safe iterators operate on a copy of the collection. Therefore, they do not throw an exception if the collection is modified during iteration. Examples: `ConcurrentHashMap`, `CopyOnWriteArrayList` are fail-safe.

440. You have a character array and a String. Which one is more secure to store sensitive data (like password, date of birth, etc.)?

It is safer to store sensitive information in a character array. In Java, `String` is immutable and is stored in the String pool. Once a `String` is created, it stays in the pool in memory until it is garbage collected. You have no control over garbage collection. Therefore, anyone having access to a memory dump can potentially extract the sensitive data and use it. If you use a mutable object like a character array to store the value, you can set it to blank once you are done with it. Once it is made blank, it cannot be used by anyone else.

441. Why do you use volatile keyword in Java?

The `volatile` keyword guarantees global ordering on reads and writes to a variable. This implies that every thread accessing a volatile field will read the variable's current value instead of using a cached value. By marking the variable `volatile`, the value of a variable is never cached thread-locally. All reads and writes will go straight to main memory in Java.

442. What is the difference between `poll()` and `remove()` methods of Queue in Java?

Both `poll()` and `remove()` methods remove and return the head of the Queue. When Queue is empty, `poll()` method fails and returns null, but `remove()` method fails and throws an `Exception`.

443. Can you catch an exception thrown by another thread in Java?

Yes, it can be done by using `Thread.UncaughtExceptionHandler`. Java Documentation says: "When a thread is about to terminate due to an uncaught exception, the Java Virtual Machine will query the thread for its `UncaughtExceptionHandler`

using `Thread.getUncaughtExceptionHandler`, and will invoke the handler's `uncaughtException` method, passing the thread and the exception as arguments.”

444. How do you decide which type of Inner Class – Static or Non-Static to use in Java?

An inner class has full access to the fields and methods of the enclosing class. This is convenient for event handlers, but comes at a cost. Every instance of an inner class retains and requires a reference to its enclosing class. Due to this cost, there are many situations where static nested classes are preferred over inner classes. When instances of the nested class outlive instances of the enclosing class, the nested class should be static to prevent memory leaks. At times, due to their “hidden” reference to enclosing class, Inner classes are harder to construct via reflection.

445. What are the different types of Classloaders in Java?

Java Classloader is the part of the Java Runtime Environment (JRE) that loads classes on demand into the Java Virtual Machine (JVM). When the JVM is started, three types of class loaders are used:

- Bootstrap Classloader: It loads core Java API file `rt.jar` classes from the folder.
- Extension Classloader: It loads jar files from `lib/ext` folder.
- System/Application Classloader: It loads jar files from the path specified in the `CLASSPATH` environment variable.

Classes may be loaded from the local file system, a remote file system, or even the web.

446. What are the situations in which you choose HashSet or TreeSet?

`HashSet` is better than `TreeSet` in almost every way. It gives $O(1)$ for `add()`, `remove()`, and `contains()` operations. `TreeSet` gives $O(\log(N))$ for these operations. Still, `TreeSet` is useful when you wish to maintain order over the inserted elements or query for a range of elements within the set. Use `TreeSet` when you want to maintain order or when there are enough read operations to offset the increased cost of write operations.

447. What is the use of method references in Java?

Java 8 introduced Method references. It allows constructors and methods to be used as lambdas. The main uses of Method reference are to improve code organization, clarity, and terseness.

448. Do you think Java Enums are more powerful than integer constants?

Yes. Java Enums provide many features that integer constants cannot. Enums can be considered as final classes with a fixed number of instances. Enums can implement interfaces but cannot extend another class. While implementing the strategy pattern, you can use this feature of Enums, especially when the number of strategies is fixed. You can also attach meta-data to enum values in Java. Also, enum values are typesafe, whereas integer constants are not. You can define custom behavior in enum values.

449. Why do we use static initializers in Java?

In Java, a static initializer can run code during the initial loading of a class and it guarantees that this code will only run once. Also, the static code will finish running before a class can be accessed in any way. Initializing static members from constructors is more work. You have to make sure that every constructor does this. You need to maintain a flag to mark the static work when it is done. You may have to think about synchronization or race conditions for work in the static block not initialized from the static context.

450. Your client is complaining that your code is throwing `NoClassDefFoundError` or `NoSuchMethodError`, even though you are able to compile your code without error and method exists in your code. What could be the reason behind this?

Sometimes we upgrade our libraries even with the same method name, but forget to let the client know about the new version. Due to this difference in version, we get `NoClassDefFoundError` or `NoSuchMethodError` at runtime when one library was not compatible with such an upgrade. Java build tools and IDEs can also produce dependency reports that tell you which libraries depend on that JAR. Mostly, identifying and upgrading the library that depends on the older JAR resolves the issue.

451. How can you check if a String is a number by using regular expression?

Regex is a powerful tool for matching patterns and searching patterns. A numeric String can only contain digits i.e. 0 to 9. It can also contain + and - sign at start of the String. We can create a regular expression for these two rules. One simple example is as follows:

```
Pattern pattern = Pattern.compile(".*\\D.*");
```

452. What is the difference between the expressions `String s = "Temporary"` and `String s = new String("Temporary")`? Which one is better and more efficient?

In general, `String s = "Temporary"` is more efficient to use than `String s = new String("Temporary")`. In case of `String s = "Temporary"`, a String with the value "Temporary" is created in String pool. If another String with the same value is created (e.g., `String s2 = "Temporary"`), it will reference the same object in the String pool. But, when you use `String s = new String("Temporary")`, Java creates a String with the value "Temporary" in the String pool. Also, that String object is then passed to the constructor of the String Object i.e. `new String("Temporary")`. And this call creates another String object (not in the String pool) with that value. Therefore, each such call creates an additional String object. E.g. `String s2 = new String("Temporary")` creates an extra String object, rather than just reusing the same String object from the String pool. So `String s = "Temporary"` is always an efficient way.

453. In Java, can two equal objects have the different hash code?

No. It is not possible for two equal objects to have different hashcode. But two objects with same hashcode may or may not be equal.

454. How can we print an Array in Java?

We can print an array by using methods of Arrays class. We can either use `Arrays.toString()` method or we can use `Arrays.deepToString()` method. Since array doesn't implement `toString()` method by itself, just passing an array to `System.out.println()` will not print its contents. But we can use `Arrays.toString()` to print each element of an array.

455. Is it ok to use random numbers in the implementation of `hashCode()` method in Java?

No. The hashcode of an object should be always same. If you use random number in `hashCode()` method, then you may get a different value of hashcode for same object. This will break the hashcode contract.

456. Between two types of dependency injections, constructor injection and setter dependency injection, which one is better?

Constructor injection guarantees that a class will be initialized with all its dependencies during creation. But setter injection provides flexibility to set an optional dependency. If we are using an XML file to describe dependencies, the setter injection is more readable. In general, it is a good practice to use constructor injection for mandatory dependencies and use setter injection for optional dependencies.

457. What is the difference between DOM and SAX parser in Java?

In Java, Document Object Model (DOM) parser loads the whole XML into memory and creates a tree based on DOM model. This helps it in quickly locating the nodes, and making a change in the structure of XML. On the other hand, Simple API for XML (SAX) parser is an event based parser. It doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and is not suitable to parse large XML files.

458. Between Enumeration and Iterator, which one has better performance in Java?

- Enumeration interface is a read-only interface. It has better performance than Iterator. It is almost twice as fast as compared to an Iterator. It also uses very less memory. Also Enumeration does not have `remove()` method.
- On the other hand, Iterator interface is safer than Enumeration, since it can check whether a collection is modified or not during iteration. If a collection is altered while an Iterator is iterating, then it throws `ConcurrentModificationException`.

459. What is the difference between pass by reference and pass by value?

Whenever an object is passed by value, it means that a copy of the object is passed. Even if changes are made to that object, it doesn't affect the original value. Whenever an object is passed by reference, it means that the actual object is

not passed, rather a reference of the object is passed. Therefore, any changes made by an external method, are also reflected in the actual object and its reference.

460. What are the different ways to sort a collection in Java?

The most popular way to sort a collection in Java is by calling `Collections.sort()` method. You can provide your custom `Comparator` to `sort()` method for sorting the data in your custom way. The other way is to use a Sorted collection like `TreeSet` or `TreeMap` that stores the information in a sorted order and then you can convert it to a `List`.

461. Why Collection interface doesn't extend Cloneable and Serializable interfaces?

Collection interface just specifies groups of objects known as elements. Each concrete implementation of a Collection can choose its own way of how to maintain and order its elements. Some collections may allow duplicate keys, while other collections may not. A lot of collection implementations have clone method. But many do not. It is not worthwhile to include it in all, since Collection is an abstract representation. What matters is the concrete implementation. Cloning and serialization come into picture while doing concrete implementation. Therefore, the concrete implementations of collections should decide how they can be cloned or serialized.

462. What is the difference between a process and a thread in Java?

A process is simply an execution of a program. A Thread is a single execution sequence within a process. A process may contain multiple threads. A Thread is also called as a lightweight process.

463. What are the benefits of using an unordered array over an ordered array?

In an ordered array the search time has time complexity of $O(\log n)$. Whereas, in an unordered array, search time complexity is $O(n)$. In an ordered array, the insert operation has a time complexity of $O(n)$. Whereas, the insertion operation for an unordered array takes constant time of $O(1)$. Therefore, when we have more writes than reads, it is preferable to use an unordered array.

464. Between HashSet and TreeSet collections in Java, which one is better?

A `HashSet` is Implemented using a `HashTable`. Therefore, its elements are stored in a random order. The `add()`, `remove()`, and `contains()` methods of a `HashSet` have constant time complexity $O(1)$. A `TreeSet` is implemented using a tree data structure. The elements in a `TreeSet` are sorted in a natural order. Therefore, `add()`, `remove()`, and `contains()` methods have time complexity of $O(\log n)$. So from performance perspective, `HashSet` has better performance than `TreeSet`. But if you want to store elements in a natural sorting order, then `TreeSet` is a better collection.

465. When does JVM call the finalize() method?

JVM instructs the Garbage Collector to call the `finalize` method, just before releasing an object from the memory. A programmer can implement `finalize()` method to explicitly release the resources held by the object. This will help in better memory management and avoid any memory leaks.

466. When would you use Serial Garbage collector or Throughput Garbage collector in Java?

- The Serial Garbage collector is used for small applications that require heap memory upto 100 MB.
- The Throughput Garbage collector is used in medium to large size Java applications.

467. In Java, if you set an object reference to null, will the Garbage Collector immediately free the memory held by that object?

No. JVM decides to run the Garbage Collector whenever it is low on memory. When Garbage Collector runs, it looks for objects that are available for garbage collection and then frees the memory associated with this object. So just setting an Object reference null makes it eligible for Garbage Collection, but it does not immediately free the memory.

468. How can you make an Object eligible for Garbage collection in Java?

To make an Object eligible for Garbage collection, just make sure that it is unreachable to the program in which it is currently defined / created / used. You can set the object reference to null and make sure no other object refers it. Once the object cannot be reached, Garbage Collection can clean it during the next run.

469. When do you use Exception or Error in Java? What is the difference between these two?

Throwable class is the superclass of Exception and Error classes in Java. When you want to catch the exceptional conditions that your program can create or encounter, then use the Exception class or subclass of Exception. When you come across situations that are unexpected then use Error class in Java. Also recovering from Error is not possible in most of cases. So it is better to terminate the program.

470. What is the advantage of PreparedStatement over Statement class in Java?

PreparedStatements are precompiled statements for database queries. Due to this their performance is much better. Also, we can reuse PreparedStatement objects with different input values to the same query. Whereas, Statement class does not provide these features.

471. In Java, what is the difference between throw and throws keywords?

When we want to raise an exception in our code, we use the throw keyword with the name of the exception to be raised. Whereas, throws keyword is used in method declaration. Throws keyword tells us the Exception that can be thrown by this method. Any caller of this method should be prepared to expect this Exception. Another minor difference is that throw is used only with one exception, but throws can be used with comma-separated list of multiple exceptions.

472. What happens to the Exception object after the exception handling is done?

Once the exception handling is complete, the Exception object is not reachable. Then it is garbage collected in the next run of Garbage Collector.

473. How do you find which client machine is sending request to your servlet in Java?

We can use the ServletRequest class to find the IP address or host name of the client machine. There are methods getRemoteAddr() to get the IP address of the client machine and getRemoteHost() to get the host name of the client machine.

474. What is the difference between a Cookie and a Session object in Java?

Both Cookie and Session are used during communication between Client and Server. The Client can disable a Cookie. Due to which the Web server cannot send a cookie. But a client cannot disable a session. So a Session always works irrespective of any setting at the client side. Also a Session can store any Java object. But the Cookie can only store small information in a String object.

475. Which protocol does Browser and Servlet use to communicate with each other?

HTTP protocol. The Browser and Servlet communicate with each other by using the HTTP protocol.

476. What is HTTP Tunneling?

There are many network communication protocols on the Internet. But HTTP is the most popular among them. HTTP Tunneling is a technique in which HTTP or HTTPS protocol encapsulated the communication done by any other type of protocol. The masking of other protocol requests as HTTP requests is known as HTTP Tunneling.

477. Why do we use JSP instead of Servlet in Java?

Since JSP pages are dynamically compiled into servlets, the programmers can easily make updates to the presentation layer code. For better performance, JSP pages can be pre-compiled. Also JSP pages provide flexibility to combine static templates like HTML or XML snippets. In addition, programmers can make logic changes at the class level, without editing the JSP pages that use the class logic.

478. Is empty '.java' file name a valid source file name in Java?

Yes. You can create a class and store it in a file with name .java. You can try it yourself, by creating, compiling and running such a file. It will run correctly.

479. How do you implement Servlet Chaining in Java?

To implement Servlet Chaining, there has to be more than one servlet. The output of one servlet has to be sent to a second servlet. The output of the second servlet can be sent to a third servlet, and so on. In this way, a chain of servlets is formed to complete a task. The last servlet in the chain will be responsible for sending final response to client.

480. Can you instantiate this class?

```
1 public class A {  
2     A a = new A();  
3 }
```

No, this class cannot be instantiated, since it will result in recursively calling its constructor.

481. Why Java does not support operator overloading?

Java supports Method overloading but does not support operator overloading. It would make the design more complex by adding operator loading. Also it will make more complex compiler. One more reason is that, it will reduce the performance of JVM by operator overloading, since JVM has to do extra work to find the real meaning of overloaded operators at run time.

482. Why String class is Immutable or Final in Java?

Since String objects are cached in a String pool, it makes sense to make the String immutable. The cached String literals are shared between multiple clients. And there is a possibility that one client's action may affect another client's access to String pool. String is also used as a parameter in many Java classes. Eg. You can pass hostname, port number as String while opening a network connection. If any one can modify your copy of the String, it can change the hostname. Due to this reason, it makes sense to make String final as soon as it is created.

483. What is the difference between sendRedirect and forward methods?

When you use sendRedirect method, it creates a new request. When you use the forward method, it just forwards a request to a new target. In case of sendRedirect, the previous request scope objects are not available, because it creates a new request. In case of forward method, the previous request scope objects are available after forwarding. Also the sendRedirect method is considered slower than the forward method.

484. How do you fix your Serializable class, if it contains a member that is not serializable?

If you want to make a class Serializable, but find that this class contains members that are not Serializable, then you have to mark those members as transient. This will ensure that this member is not persisted to a stream of bytes during Serialization. Therefore, Transient keyword of Java comes to help in this scenario.

485. What is the use of run time polymorphism in Java?

During the run time the behavior of an Object can change based on its run time state. Due to this run time polymorphism is introduced in Java. If you override a method in a child class, then you are providing run time polymorphism. Nothing will happen at the compile time. But at the run time, JVM decides which method will be called based on the class of the Object.

486. What are the rules of method overloading and method overriding in Java?

- When we want to overload a method, we need to make sure that the method name remains same. But method signature can vary in the number or datatype of arguments or in the order of arguments.
- When we want to override a method, we ensure that the method is not throwing checked exceptions that are new or higher than those declared by the overridden method. Also we make sure that the method name, arguments and return type remain the same.
- Also we cannot override Static and Final methods in Java.

487. What is the difference between a class and an object in Java?

A Class is a template or a blueprint of an Object to be created. An Object is an instance of a Class. A Class defines the methods and member variables. But an Object populates the values of the member variables. Therefore a class is a

blueprint that you use to create objects. An object is an instance of a class – it is a concrete 'thing' that you made using a specific class. Most of the OOPS concepts are valid only when an Object is created.

488. Can we create an abstract class that extends another abstract class?

Yes. An abstract class can extend another abstract class. It does not need to define the methods of parent abstract class. Only the last non-abstract class has to define the abstract methods of a parent abstract class.

489. Why do you use Upcasting or Downcasting in Java?

When we want to cast a Sub class to Super class, we use Upcasting. It is also known as widening. Upcasting is always allowed in Java. When we want to cast a Super class to Sub class, we use Downcasting. It is also known as narrowing. At times, Downcasting can throw the ClassCastException if it fails the type check.

490. What is the reason to organize classes and interfaces in a package in Java?

As the name suggests, a package contains a collection of classes. It helps in setting the category of a file. Like- whether it is a Data Access Object (DAO) or an API. It helps in preventing the collision of Namespace. Also we can introduce access restriction by using package and the right modifiers on a class and its methods.

491. What is information hiding in Java?

Information hiding is OOPS concept. In Java you can use encapsulation to do Information hiding. An object can use the access modifiers like-public, private, protected to hide its internal details from another object. This helps in decoupling the internal logic of an object from outside world. By using Information hiding, an object can change its internal implementation without impacting the outside calling client's code.

492. Why does Java provide default constructor?

In Java all the interaction takes place between Object instances. To create an Object instance, JVM needs a constructor. Java does not enforce the rule on a programmer to define a default constructor for every class. Whenever an object has to be created and programmer has not provided a constructor, Java uses default constructor to create the object. Default constructor also initializes member variables with their default values.

493. What is the difference between super and this keywords in Java?

We use super keyword to access the methods of the super class from child class. We use this keyword to access methods of the same class.

494. What is the advantage of using Unicode characters in Java?

Unicode characters have much larger number of characters in the specification. They also contain Asian and non-western European characters. Most of the modern technologies, websites and browsers support these Unicode characters.

495. Can you override an overloaded method in Java?

Yes. Java allows to override an overloaded method, if that method is not a static or final method.

496. How can we change the heap size of a JVM?

Java provides the command line parameters to set the heap size for JVM. You can specify the values in -Xms and -Xmx parameters. These parameters stand for initial and maximum heap size of JVM.

497. Why should you define a default constructor in Java?

In general, Java provides a default constructor with each class. But there are certain cases when we want to define our own version of default constructor. When we want to construct an object with default values, we create our default constructor. At times, we can mark the default constructor private. So that any other class cannot create an instance of our class. This technique is generally used in Singleton design pattern.

498. How will you make an Object Immutable in Java?

- To make an object immutable follow these two rules. One, do not use any setter methods that can change the fields of your class. Two, make the fields final.

- By following these rules, the member variables cannot be changed after initialization. This will ensure that member variables of an Object do not change. And thus the Object will be considered Immutable.

499. How can you prevent SQL Injection in Java Code?

In Java, you can use PreparedStatement to prevent SQL injection. In a PreparedStatement you can pass the precompiled SQL queries with pre-defined parameters. This helps in checking the type of parameters to SQL queries. So it protects your code from SQL injection attacks.

500. Which two methods should be always implemented by HashMap key Object?

Any object that we want to use as key for HashMap or in any other hash based collection data structure e.g. Hashtable, or ConcurrentHashMap must implement equals() and hashCode() method.

501. Why an Object used as Key in HashMap should be Immutable?

The Key object should be immutable so that the hashCode() method always returns the same value for that object. The hash code returned by hashCode() depends on the values of the member variables of an object. If an object is mutable, then the member variables can change. Once the member variables change, the hash code changes. If the same object returns different hash codes at different times, then it is not reliable to be used in the HashMap. For example, when you insert the object, the hash code is X, so the HashMap will store it in bucket X. But when you search for it, the hash code is Y, so HashMap will look for the object in bucket Y. As a result, you are not getting what you stored. To solve this, a key object should be immutable. Although the compiler does not enforce this rule, a good programmer always remembers this rule.

502. How can we share an object between multiple threads?

There are many ways to share the same object between multiple threads. You can use a BlockingQueue to pass an object from one thread to another. You can also use the Exchanger class for this purpose. An Exchanger is a bidirectional form of a SynchronousQueue in Java, and you can use it to swap objects as well.

503. How can you determine if your program has a deadlock?

If we suspect that our application is stuck due to a deadlock, we can take a thread dump by using the command specific to the environment in which the application is running. For example, in Linux you can use the command kill -3. In case of a deadlock, you will see in the thread dump the current status and stack trace of threads in the JVM, and one or more of them will be stuck with the message "deadlock." You can also do this programmatically by using the ThreadMXBean class that ships with the JDK. If you don't need programmatic detection, you can use JConsole. On the thread tab, there is a "detect deadlock" button.

1. What are Wrapper classes in Java?

Java provides Wrapper classes to allow primitive types to be accessed as objects. Primitive types like boolean, int, double, float, etc., have corresponding Wrapper classes – Boolean, Integer, Double, Float, etc. Many of these Wrapper classes are in the java.lang package. Java 5.0 introduced Autoboxing and Unboxing for Wrapper classes.

```
1 public class WrapperTest {
2     public static void main(String[] args) {
3         // Converting int into Integer
4         int count = 50;
5
6         Integer i = Integer.valueOf(count); // Explicit conversion
7         Integer j = count; // Autoboxing (compiler converts to Integer.valueOf(count))
8
9         System.out.println(count + " " + i + " " + j);
10    }
11 }
12
```

2. What is the purpose of native method in Java?

The native keyword is used to indicate that a method is implemented in native code using JNI (Java Native Interface). Native methods allow Java developers to directly access platform-specific APIs and are often linked to native libraries.

3. What is System class?

System.class is a final class provided by the java.lang package. It contains several useful class fields and methods. The purpose of the System class is to provide access to system resources.

4. What is System, out and println in System.out.println method call?

System is a final class provided by the java.lang package. out refers to the PrintStream class and is a static member of the System class. println is a method of the PrintStream class.

5. What is the other name of Shallow Copy in Java?

Object Cloning. A Shallow Copy just copies the values of references in a class.

6. What is the difference between Shallow Copy and Deep Copy in Java?

A Shallow copy just copies the values of the references in the class. A Deep copy copies the values of the objects as well.

7. What is a Singleton class?

A Singleton class in Java has a maximum of one instance present in the JVM at all times. The constructor is written so that it never creates more than one object of the same class.

8. What is the difference between Singleton class and Static class?

A static class in Java has only static methods and acts as a container of functions, created based on procedural programming design. A Singleton class is a pattern in Object Oriented Design and has only one instance in the JVM. This pattern ensures only one instance is present at any time.

9. What are the implicit objects in JSP?

- Request
- Response
- Application
- Exception
- Page
- Config
- Session

10. How will you extend JSP code?

We can extend JSP code by using Tag libraries and Custom actions.

11. How will you handle runtime exceptions in JSP?

We use the Errorpage attribute in JSP to catch runtime exceptions. This attribute forwards the user request to the error page automatically.

12. How will you prevent multiple submits of a page that come by clicking refresh button multiple times?

We can use the Post Redirect Get (PRG) pattern to solve the issue of multiple submissions of the same data. It works as follows: When a user submits a form to the server via POST or GET, we update the state in the application database. Then we send a redirect response to the client. We then load a view using GET; since this is a new JSP page, it is safe from multiple submits. The code that processes the request is idempotent, so it does not perform the same action twice for the same request.

13. How will you implement a thread safe JSP page?

We can use the SingleThreadModel Interface to implement a thread-safe JSP page. Alternatively, we can add the directive in the JSP page to make it thread safe.

14. How will you include a static file in a JSP page?

We can use the include directive of JSP to include a static page. This approach uses the translation phase to include the static page. Specify the URL of the resource to be included as the file attribute in this directive.

15. What are the lifecycle methods of a JSP?

- `jspInit()`: This method is invoked when the JSP is called for the first time. It is used for initial setup for servicing a request.
- `_jspService()`: This method is used to serve every request of the JSP.
- `jspDestroy()`: When a JSP is removed from the container, this method is called for cleanup of resources like database connections.

16. What are the advantages of using JSP in web architecture?

- Performance: JSP provides good performance due to its design for servicing multiple requests with the same code.
- Fast: Since JSP is pre-compiled, the server can serve pages very quickly.
- Extendable: JSP is based on Java Servlets, which helps in extending JSP architecture with other Java technologies like JDBC, JMS, JNDI, etc.
- Design: It is easier to design user interfaces with JSP since it is very close to HTML. UI designers can create a JSP with mock data, and developers can later implement dynamic data.

17. What is the advantage of JSP over Javascript?

In JSP, we can write Java code seamlessly, allowing interaction with the rest of the application. JavaScript code is mostly executed on the client side, limiting its capabilities; for example, it cannot connect to a database server from the client side.

18. What is the Lifecycle of JSP?

- Compilation: When a request is made for a JSP, the corresponding JSP is converted into a Servlet and compiled. If a compiled form already exists and there is no change since last compilation, this stage does nothing.
- Initialization: In this stage, the `jspInit()` method is called to initialize any data or code to be used multiple times in `_jspService()`.
- Service: For each request to the JSP, `_jspService()` is called to service the request. This is the core logic that generates responses.
- Destroy: When the JSP is removed from the container/server, resources are cleaned up just before removal.

19. What is a JSP expression?

A JSP expression is an element used to evaluate a Java expression and convert it into a String, which is then inserted into the JSP page wherever the expression occurs.

20. What are the different types of directive tags in JSP?

- Page: Used for page-related attributes; typically placed at the top of the page.
- Taglib: Allows creation and use of custom tags.
- Include: Reads a file and merges its content with the JSP during compilation.

21. What is session attribute in JSP?

The session attribute in JSP is used for the HTTP session mechanism. If you do not want to use HTTP session in JSP, set this attribute to false. If set to true, you can use the built-in session object.

22. What are the different scopes of a JSP object?

- Page: The object is accessible from the page where it was created; objects are recreated when the page is refreshed.
- Request: The object is accessible to the HTTP request that created it.
- Session: The object is available throughout the same HTTP session.
- Application: The object is available throughout the application in which the JSP was created.

23. What is pageContext in JSP?

In JSP, `pageContext` is an implicit object used for storing and accessing all page scope objects. It is an instance of the `PageContext` class from the `javax.servlet.jsp` package.

24. What is the use of `jsp:useBean` in JSP?

We use `jsp:useBean` to invoke the methods of a Java Bean class. The Java Bean class holds data and provides setter/getter methods. The container locates or creates the bean instance, which can then be used in expressions or JSP code.

25. What is difference between `include` Directive and `include` Action of JSP?

- Include directive is called at translation phase to include content in JSP; include Action is executed during runtime.
- Parameters cannot be passed to include directive, but include action can accept parameters via `jsp:param` tag.
- Include directive copies content from another file for compilation; include action dynamically processes and includes the resource at runtime.

26. How will you use other Java files of your application in JSP code?

We can use the `import` tag to import a Java file in JSP code. Once imported, JSP code can use the classes. It's recommended to organize Java code in packages for better use in JSP.

27. How will you use an existing class and extend it to use in the JSP?

We can use the `extends` attribute in the `include` tag to extend an existing class in the current JSP.

28. Why `_jspService` method starts with `_` symbol in JSP?

All code written in a JSP is placed in the `_jspService` method during the translation phase. This method cannot be overridden, while other lifecycle methods like `jspInit()` and `jspDestroy()` can be overridden. The container uses the `_` symbol to distinguish methods that cannot be overridden by client code.

29. Why do we use tag library in JSP?

Taglib is useful for creating UI frameworks with custom tags. It allows UI designers to use custom tags in HTML without knowledge of the underlying implementation, and promotes code reusability.

30. What is the different type of tag library groups in JSTL?

- Core tags
- SQL tags
- Formatting tags
- XML tags
- JSTL Functions

31. How will you pass information from one JSP to another JSP?

Information can be passed using implicit objects. For JSPs called in the same session, use the session object. For included JSPs, use `jsp:param` to pass information.

32. How will you call a stored procedure from JSP?

JSP allows running Java code. A stored procedure can be called using JDBC code, specifically through a `CallableStatement`. If using the Spring framework, `JdbcTemplate` can be used to invoke stored procedures.

33. Can we override `_jspService()` method in JSP?

No, JSP specification does not allow overriding of the `_jspService` method. Other methods like `jspInit()` and `jspDestroy()` can be overridden.

34. What is a directive in JSP?

A JSP directive is a way to pass messages to the JSP container. It does not produce output to the page but communicates with the container. Directives may have attributes for passing additional information. Important directives include `page`, `include`, and `taglib`.

35. How will you implement Session tracking in JSP?

- Cookies: Set session information in a cookie and use it for subsequent requests.

- Hidden Form Field: Send session id in a hidden HTML form field.
- Session object: Use the built-in session object in JSP.
- URL Rewriting: Add session id at the end of a URL (e.g., www.abcserver.com?sessionId=1234).

36. How do you debug code in JSP?

The simplest method is to write logger statements or `System.out.println()` to log files. When the JSP is called, log messages are written to logs, aiding debugging. Alternatively, link the JSP container with an IDE for debugging with breakpoints and step-through operations.

37. How will you implement error page in JSP?

Create a JSP with error page handling information and a user-friendly message, logging important information like stack trace. Add the parameter `isErrorPage=true` in the page directive to identify the error page.

Use this error page in other JSPs to handle errors by directing them to the error page.

38. How will you send XML data from a JSP?

Set `contentType="text/xml"` in the page directive to send data in XML format.

39. What happens when we request for a JSP page from web browser?

When a user requests a JSP page, the request goes to the web server, which forwards it to the JSP container (e.g., Tomcat). If the JSP has not been compiled before, the container translates it into a servlet and compiles it. Then, the JSP code is loaded into memory, and `jspInit()` and `_jspService()` methods are called. The `_jspService()` method generates the output sent to the client browser.

40. How will you implement Auto Refresh of page in JSP?

Use the `setIntHeader()` method to set the refresh frequency. Send the key "Refresh" with the time in seconds for auto refresh.

```
response.setIntHeader("Refresh", 10)
```

41. What are the important status codes in HTTP?

- 200: Request is successful.
- 400: Bad request.
- 401: Request not authorized.
- 404: Resource not found.
- 503: Service not available.

42. What is the meaning of Accept attribute in HTTP header?

In an HTTP header, the Accept attribute specifies the MIME types that a client or browser can handle. MIME type is an identifier for the type of file or data being sent over the internet.

43. What is the difference between Expression and Scriptlet in JSP?

An Expression in JSP returns a value and displays it at a specific location, typically used for dynamic information such as time or counters. A Scriptlet allows writing Java code in JSP, defining variables, methods, and handling more complex tasks. Scriptlets can be reused.

44. How will you delete a Cookie in JSP?

- `setMaxAge()`: Set the maximum age of a cookie; after this period, it expires and is deleted.
- Header: Set the expiry time in the response header using `Response.setHeader()`, which expires the cookie after the specified time.

45. How will you use a Cookie in JSP?

First, create a Cookie object and set its name and value. Set the expiry time using `setMaxAge()`. Send the cookie in the HTTP response header so it is stored in the client browser until its maximum age. To retrieve it, use `getCookies()` to get all cookies on the client and iterate through them to find the required value.

46. What is the main difference between a Session and Cookie in JSP?

A Session is stored on the server side and is a built-in object in the JSP container. A Cookie is stored on the client side. Both methods can be used for session tracking, but storing cookies requires user permission.

47. How will you prevent creation of session in JSP?

Set the session attribute as false in the page directive to prevent creation of the session object.

48. What is an output comment in JSP?

Output in JSP can be formatted as a comment in HTML code. Such comments are not visible in the browser but can be seen when viewing the page source.

49. How will you prevent caching of HTML output by web browser in JSP?

Set the header in the response object for Cache-Control to specify no caching.

```
response.setHeader("Cache-Control", "no-store");
```

```
response.setDateHeader("Expires", "0");
```

50. How will you redirect request to another page in browser in JSP code?

Use `sendRedirect()` in JSP to redirect the request to another location or page. The request will redirect in the browser itself and not return to the server.

51. What is the difference between `sendRedirect` and `forward` in a JSP?

Both `forward` and `sendRedirect` are mechanisms for sending a client to another page. The main differences are as follows:

- In `forward`, the processing takes place on the server side. In the case of `sendRedirect()`, the processing happens on the client side.
- In `forward`, the request is transferred to another resource within the same server. In `sendRedirect`, the request can be transferred to a resource on another server.
- In `forward`, only one request call is consumed. In `sendRedirect`, two request-response calls are created and consumed.
- `Forward` is declared in the `RequestDispatcher` interface. `sendRedirect` is declared in the `HttpServletResponse` object.

52. What is the use of `config` implicit object in JSP?

In JSP, the `config` object is of type `ServletConfig`. This object is created by the Servlet Container for each JSP page and is used for setting initialization parameters for a specific JSP page.

53. What is the difference between `init-param` and `context-param`?

Both `init-param` and `context-param` can be specified in the `web.xml` file. `init-param` is used to specify parameters that are specific to a servlet or JSP; this information is confined to the scope of that JSP. `context-param` is used to specify parameters for the overall application scope; this information does not change easily and can be used by all JSP/Servlets in that Container.

54. What is the purpose of `RequestDispatcher`?

The `RequestDispatcher` interface is used to forward requests to other resources like HTML, JSP, etc. It can also be used to include the content of another page in a JSP. It has two methods: `forward` and `include`. You must first get the `RequestDispatcher` object from the container, then call the `include` or `forward` method on this object.

55. How can you read data from a Form in a JSP?

JSP provides a built-in request object with methods to read form data:

- `getParameterNames()`: Returns the list of all the parameters in the form.
- `getParameter()`: Gets the value of a parameter set in the form. Returns null if the parameter is not found.
- `getParameterValues()`: If a parameter is mentioned multiple times in a form, use `request.getParameterValues()` to get all the values. This method returns an array of String values.

- `getParameterMap()`: Returns a map of all the parameters in the form.

56. What is a filter in JSP?

Filters in JSP are used to intercept requests from a client or to modify responses from a server. A filter is a Java class defined in the deployment descriptor (`web.xml`) of an application. The JSP container reads filters from `web.xml` and applies them as per the URL pattern associated with the filter. The JSP engine loads all filters when the server starts.

57. How can you upload a large file in JSP?

To upload a file in JSP, use `<input type="file">` in the HTML form. For large files, set `enctype="multipart/form-data"` and use the POST method. Once the request is received, you can implement logic to read multipart data in the `doPost()` method of JSP. JSP frameworks provide methods to read large files in this way.

58. In which scenario does the Container initialize multiple JSP/Servlet objects?

To initialize multiple JSP objects, specify the same Servlet object multiple times in `web.xml`. This tells the JSP container to initialize separate JSP/Servlet objects for each element. Each Servlet instance will have its own `ServletConfig` object and parameters.

59. When will you use Strategy Design Pattern in Java?

The Strategy pattern is useful for implementing a family of algorithms and is a behavioral design pattern. With the Strategy pattern, you can select the algorithm at runtime, such as choosing a sorting strategy for data or saving files in different formats (`.txt`, `.csv`, `.jpg`). You create an interface as an abstraction for clients, then provide multiple implementations with different algorithms. At runtime, you can vary the algorithm based on the type of request. The Strategy pattern hides algorithm implementation details from the client. The `Java Collections.sort()` method uses the strategy design pattern.

60. What is Observer design pattern?

In the Observer design pattern, a Subject maintains a list of Observers waiting for updates. When the Subject changes, it notifies all observers. For example, students waiting for test results are observers, and the test is the subject. Once results are available, the testing organization notifies all students. The Observer pattern is popular in the MVC architectural pattern. A main issue is possible memory leaks if observers are not de-registered in time because the subject holds strong references to observers.

61. What are the examples of Observer design pattern in JDK?

Examples of Observer pattern in JDK include:

- `java.util.Observer`, `java.util.Observable`
- `javax.servlet.http.HttpSessionAttributeListener`
- `javax.servlet.http.HttpSessionBindingListener`
- All implementations of `java.util.EventListener`, also in Swing packages
- `javax.faces.event.PhaseListener`

62. How is Strategy design pattern different from State design pattern in Java?

State design pattern defines a state machine for an object, where each state is a subclass of State. Actions change the object's state. Strategy pattern defines multiple algorithms; the client can change the algorithm used for the same action. While both can swap implementations, State pattern chooses based on the object's internal state, whereas Strategy pattern uses external parameters from the client. State pattern is useful for maintainability in large codebases.

63. Can you explain Decorator design pattern with an example in Java?

The Decorator (or Wrapper) pattern adds behavior to an object without changing the behavior of other objects of the same class. In `java.io`, you can decorate a `FileInputStream` with `BufferedInputStream`, then `GzipInputStream`, and finally `ObjectInputStream` for serialization. Each step adds new behavior:

```
FileInputStream fis = new FileInputStream("/myfile.gz");
BufferedInputStream bis = new BufferedInputStream(fis);
```

```
GzipInputStream gis = new GzipInputStream(bis);
```

```
ObjectInputStream ois = new ObjectInputStream(gis);
```

Here, `FileInputStream` is successively decorated with additional features.

64. What is a good scenario for using Composite design Pattern in Java?

Composite pattern is ideal for tree structures (like organizational hierarchies), recursion-based algorithms, and graphics. For example, an organization can be modeled as a tree where managers and employees are nodes. Composite is also useful in recursive algorithms, such as polynomial solving, and in graphics, where shapes are grouped into composites for rendering.

65. Have you used Singleton design pattern in your Java project?

Yes, Singleton is widely used in enterprise Java applications. It ensures that only one instance of a class exists throughout the application, which is helpful for storing properties or configuration data in a single location.

66. What are the main uses of Singleton design pattern in Java projects?

Main uses include:

- Runtime: `java.lang.Runtime` is a singleton; only one instance per JVM interfaces with the environment.
- Enum: Java enums are based on Singleton pattern for globally accessible values.
- Properties: Storing configuration properties in a Singleton ensures consistency across classes.
- Spring: Beans are Singleton by default per container; only one instance exists. The scope can be changed to prototype if needed.

67. Why is `java.lang.Runtime` a Singleton in Java?

`java.lang.Runtime` acts as an interface to the environment in which the Java process is running. It has methods like `totalMemory()`, `maxMemory()`, `exit()`, and `gc()`. To prevent multiple objects from controlling these resources, only one instance of `Runtime` exists, implemented as a Singleton.

68. What is the way to implement a thread-safe Singleton design pattern in Java?

Thread-safe Singleton can be implemented in multiple ways:

Double Checked Locking:

```
2  class DoubleCheckSingleton {
3      // volatile prevents instruction reordering and ensures visibility across threads
4      private volatile HelloSingleton helloSingleton;
5
6      public HelloSingleton getHelloSingleton() {
7          HelloSingleton result = helloSingleton; // read into local for performance
8          if (result == null) {
9              synchronized (this) {
10                 result = helloSingleton;
11                 if (result == null) {
12                     result = new HelloSingleton();
13                     helloSingleton = result;
14                 }
15             }
16         }
17         return result;
18     }
19 }
20
21 // Example payload class
22 class HelloSingleton {
23     // fields/methods as needed
24 }
```

69. What are the examples of Singleton design pattern in JDK?

Examples include:

- `java.lang.Runtime.getRuntime()`: Returns the single `Runtime` instance per JVM.
- `java.lang.System.getSecurityManager()`: Returns the `SecurityManager` for the platform.

- `java.awt.Desktop.getDesktop()`

70. What is Template Method design pattern in Java?

Template Method is a behavioral design pattern for creating the outline of an algorithm or operation. The skeleton is in a base class, and steps are delegated to subclasses for customization. For example, in a game, methods like `initializeGame()`, `makeMove()`, `endGame()` can be abstract in the base class and implemented differently in subclasses (e.g., Monopoly or Chess). This provides customizable classes for users.

71. What are the examples of Template method design pattern in JDK?

Examples include:

- Java Abstract Collection classes like `java.util.AbstractList`, `java.util.AbstractSet`, and `java.util.AbstractMap` provide templates for their collections.
- `javax.servlet.http.HttpServlet`: `doGet()`, `doPost()`, etc., send HTTP 405 error responses as templates that can be customized.
- `java.io.Stream` and `Writer` classes like `InputStream`, `OutputStream`, `Reader`, and `Writer` provide non-abstract methods as templates.

72. Can you tell some examples of Factory Method design pattern implementation in Java?

Factory Method pattern is a creational design pattern. Examples in JDK:

- `Java.lang.Class.forName()`
- `java.net.URLStreamHandlerFactory.createURLStreamHand`
- `java.util.Calendar.getInstance()`
- `java.util.ResourceBundle.getBundle()`
- `java.text.NumberFormat.getInstance()`
- `java.nio.charset.Charset.forName()`
- `java.util.EnumSet.of()`
- `javax.xml.bind.JAXBContext.createMarshaller()`

73. What is the benefit we get by using static factory method to create object?

Static factory methods encapsulate the creation process of an object. Instead of direct constructor calls, a static method creates the object, allowing the Factory to choose the correct implementation at runtime. This can provide the desired behavior and manage resources, such as limiting the number of connections in a `ConnectionPool`.

74. What are the examples of Builder design pattern in JDK?

Examples include:

- `java.lang.StringBuilder.append()`: `StringBuilder` uses the Builder pattern.
- `java.nio.IntBuffer.put()`: Methods for building `IntBuffer`.
- `javax.swing.GroupLayout.Group.addComponent()`: Builds UI components.
- `java.lang.Appendable`
- `java.lang.StringBuffer.append()`: Similar to `StringBuilder`, uses Builder pattern.

75. What are the examples of Abstract Factory design pattern in JDK?

Examples include:

- `javax.xml.xpath.XPathFactory.newInstance()`
- `javax.xml.parsers.DocumentBuilderFactory.newInstance()`
- `javax.xml.transform.TransformerFactory.newInstance()`

76. What are the examples of Decorator design pattern in JDK?

Examples include:

- java.io package: InputStream, OutputStream, Reader, and Writer subclasses can decorate instances with additional behavior.
- java.util.Collections: Methods like checkedCollection(), checkedList(), checkedMap(), synchronizedList(), synchronizedMap(), synchronizedSet(), unmodifiableSet(), unmodifiableMap(), unmodifiableList() decorate and return the same type.
- javax.servlet package: HttpServletRequestWrapper and HttpServletResponseWrapper use the Decorator pattern.

77. What are the examples of Proxy design pattern in JDK?

Examples include:

- java.lang.reflect.Proxy
- java.rmi.*
- javax.inject.Inject
- javax.ejb.EJB
- javax.persistence.PersistenceContext

78. What are the examples of Chain of Responsibility design pattern in JDK?

Examples include:

- java.util.logging.Logger.log(): Logger class has multiple log() methods for different scenarios; the client calls the appropriate method and Logger handles the command.
- javax.servlet.Filter.doFilter(): In the Filter class, the container calls doFilter when a request/response pair passes through the chain. FilterChain allows passing the request and response to the next level.

79. What are the main uses of Command design pattern?

The Command pattern encapsulates all information needed to trigger an event. Main uses include:

- Graphic User Interface (GUI): Used in menu items and buttons to trigger actions.
- Macro Recording: User actions implemented as commands can be recorded and replayed.
- Multi-step Undo: Each step as a Command enables Undo functionality.
- Networking: Commands can be sent over the network to execute actions remotely.
- Progress Bar: Installation routines as series of Commands can display progress.
- Wizard: Steps in a wizard flow can be implemented as Commands.
- Transactions: Transactional code can use Commands for each task or update.

80. What are the examples of Command design pattern in JDK?

Examples include:

- All implementations of java.lang.Runnable
- All implementations of javax.swing.Action

81. What are the examples of Interpreter design pattern in JDK?

Interpreter pattern is used to evaluate sentences in a language, such as SQL. Each keyword or symbol is represented by a class, and the sentence is a composite syntax tree. Examples in JDK include:

- java.util.Pattern
- java.text.Normalizer
- Subclasses of java.text.Format: DateFormat, MessageFormat, NumberFormat
- Subclasses of javax.el.ELResolver: ArrayELResolver, MapELResolver, CompositeELResolver

82. What are the examples of Mediator design pattern in JDK?

Mediator pattern decouples objects that interact. Examples include:

- java.util.Timer: schedule() methods act as Mediator between clients and TimerTask.
- java.util.concurrent.Executor.execute(): Acts as Mediator for executing tasks.

- `java.util.concurrent.ExecutorService`
- `java.lang.reflect.Method.invoke()`: Acts as Mediator in reflection.
- `java.util.concurrent.ScheduledExecutorService`: `schedule()` methods are Mediator pattern implementations.

83. What are the examples of Strategy design pattern in JDK?

Examples include:

- `java.util.Comparator`: `compare()` method changes the strategy used by `Collections.sort()`.
- `javax.servlet.http.HttpServlet`: `service()`, `doGet()`, `doPost()`, etc., process requests based on the selected strategy.

84. What are the examples of Visitor design pattern in JDK?

Visitor pattern allows adding new virtual methods to existing classes without modifying their structure. Examples include:

- `javax.lang.model.element.AnnotationValue` and `AnnotationValueVisitor`
- `java.nio.file.FileVisitor` and `SimpleFileVisitor`
- `javax.lang.model.type.TypeMirror` and `TypeVisitor`
- `javax.lang.model.element.Element` and `ElementVisitor`
- `javax.faces.component.visit.VisitContext` and `VisitCallback`

85. How is Decorator design pattern different from Proxy pattern?

Main differences:

- Decorator provides an enhanced interface with additional features; Proxy provides the same interface and acts as an access point.
- Decorator is a type of Composite pattern with only one component, adding features without aggregation; Proxy can improve performance via lazy loading.
- Decorator uses recursive composition; Proxy is a single object providing access to another.
- Decorator is used for building a variety of objects; Proxy is mainly used for access control.

86. What are the different scenarios to use Setter and Constructor based injection in Dependency Injection (DI) design pattern?

Setter injection is used for optional dependencies, while constructor injection is for mandatory dependencies. In Spring IoC, constructor injection is used when an object must have dependencies at creation, while setter injection is used for dependencies that can be set later or are optional.

87. What are the different scenarios for using Proxy design pattern?

Proxy pattern can be used in various scenarios:

- Virtual Proxy: Acts as a proxy for expensive-to-create objects, used in lazy loading.
- Remote Proxy: Local object providing access to a remote object (e.g., RMI, RPC).
- Protective Proxy: Controls access to a master object, authenticating and authorizing clients.
- Smart Proxy: Adds extra information such as tracking access or denying requests above a threshold.

88. What is the main difference between Adapter and Proxy design pattern?

Adapter pattern provides a different interface to an object. But the Proxy always provides the same interface to the object. Adapter is like providing an interface suitable to the client's use. But Proxy is the same interface that has an additional feature or check. For example, in electrical appliances we use an Adapter to convert from one type of socket to another type of socket. In case of a proxy, we have a plug with a built-in surge protector. The interface for the plug and the original device remains the same.

89. When will you use Adapter design pattern in Java?

If we have two classes with incompatible interfaces, we use the Adapter pattern to make it work. We create an Adapter object that can adapt the interface of one class to another class. It is generally used for working with third-party libraries.

We create an Adapter class between third-party code and our class. In case of any change in third-party code, we have to just change the Adapter code. The rest of our code can remain the same and just talk to the Adapter.

90. What are the examples of Adapter design pattern in JDK?

In JDK, there are many places where the Adapter design pattern is used. Some of these are as follows:

- `java.util.Arrays.asList()`: This method can adapt an Array to work as a List.
- `java.util.Collections.list()`: This method can adapt any collection to provide List behavior.
- `java.util.Collections.enumeration()`: This method returns an enumeration over the collection.
- `java.io.InputStreamReader(InputStream)`: This method adapts a Stream to the Reader class.
- `java.io.OutputStreamWriter(OutputStream)`: This method adapts an OutputStream to the Writer class.
- `javax.xml.bind.annotation.adapters.XmlAdapter.marshal()`

91. What is the difference between Factory and Abstract Factory design pattern?

With Factory design pattern, we can create concrete products of a type that the Factory can manufacture. For example, if it is CarFactory, we can produce Ford, Toyota, Honda, Maserati, etc. With Abstract Factory design pattern, we create a concrete implementation of a Factory. For example, DeviceFactory can be abstract and it can give us GoogleDeviceFactory, AppleDeviceFactory, etc. With AppleDeviceFactory, we will get products like iPhone, iPad, Mac, etc. With GoogleDeviceFactory, we will get products like Nexus phone, Google Nexus tablet, Google ChromeBook, etc. So it is a subtle difference between Factory and Abstract Factory design pattern. One way to remember is that within Abstract Factory pattern, Factory pattern is already implemented.

92. What is Open/closed design principle in Software engineering?

Open/closed design principle states “software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”. The open/closed principle term was originated by Bertrand Meyer in his book Object Oriented Software Construction. As per this principle, if a module is available for extension, then it is considered open. If a module is available for use by other modules, then it is considered closed. Further, Robert C. Martin has mentioned it as O in SOLID principles of Object Oriented design. It is used in State and Strategy design patterns. The Context class is closed for modification, but new functionality can be added by writing new strategy code.

93. What is SOLID design principle?

SOLID word in SOLID design principle is an acronym for:

- S: Single responsibility. A class should have a single responsibility.
- O: Open-closed. Software entities should be open for extension but closed for modification.
- L: Liskov substitution. Objects in a program should be replaceable by subclasses of the same type without any adverse impact.
- I: Interface segregation. Multiple client-specific interfaces are preferable over a single generic interface.
- D: Dependency inversion. Programs should depend on abstract entities. They should not depend on concrete implementation of an interface.

This principle was mentioned by Robert C. Martin. These are considered five basic principles of Object Oriented design. If we follow these principles, then we can create a stable program that is easy to maintain and can be extended over time.

94. What is Builder design pattern?

Builder design pattern is a creational design pattern. We can use Builder pattern to create complex objects with multiple options. For example, when we have to create a Meal in a restaurant, we can use Builder pattern. We can keep adding options like Starter, Drink, Main Course, and Dessert to create a complete meal. When a user selects other options for Starter, Drink, Main Course, Dessert, another type of meal is created. The main feature of Builder pattern is step-by-step building of a complex object with multiple options.

95. What are the different categories of Design Patterns used in Object Oriented Design?

In Object Oriented design, mainly three categories of design patterns are used. These categories are:

Creational Design Patterns:

- Builder
- Factory Method
- Abstract Factory
- Object Pool
- Singleton
- Prototype

Structural Design Patterns:

- Adapter
- Bridge
- Façade
- Decorator
- Composite
- Flyweight
- Proxy
- Behavioral Design Patterns:
- Command
- Iterator
- Chain of Responsibility
- Observer
- State
- Strategy
- Mediator
- Interpreter

96. What is the design pattern suitable to access elements of a Collection?

We can use Iterator design pattern to access the individual elements of a Collection. In case of an ordered collection, we can get an Iterator that returns the elements in order. In Java, there are many implementations of Iterator in the Collections package. We have iterators like Spliterator, ListIterator, etc. that implement Iterator pattern.

97. How can we implement Producer Consumer design pattern in Java?

We can use BlockingQueue in Java to implement Producer Consumer design pattern. It is a concurrent design pattern.

98. What design pattern is suitable to add new features to an existing object?

We can use Decorator design pattern to add new features to an existing object. With a Decorator, we work on the same object and return the same object with more features. But the structure of the object remains the same since all the decorated versions of the object implement the same interface.

99. Which design pattern can be used when to decouple abstraction from the implementation?

We can use Bridge design pattern to detach the implementation from the abstraction. Bridge is mainly used for separation of concern in design. We can create an implementation and store it in the interface, which is an abstraction. Whereas specific implementation of other features can be done in concrete classes that implement the interface. Often, Bridge design pattern is implemented by using Adapter pattern. For example, we have a Shape interface. We want to make Square and Circle shapes. But further, we want to make RedSquare, BlackSquare shapes and GreenCircle, WhiteCircle shapes. In this case, rather than creating one hierarchy of all the shapes, we separate the Color concern from the Shape hierarchy. So we create two hierarchies. One is Shape to Square and Shape to Circle hierarchy. Another one is Color to Red, Black, Green, White hierarchy. In this way, we can create multiple types of shapes with multiple colors with Bridge design pattern.

100. Which is the design pattern used in Android applications?

Android applications predominantly use Model View Presenter design pattern.

- **Model:** This is the domain model of the Android application. It contains the business logic and business rules.
- **View:** These are the UI components in your application. These are part of the view. Also, any events on UI components are part of the view module.
- **Presenter:** This is the bridge between Model and View to control the communication. Presenter can query the model and return data to view to update it.
- For example, if we have a Model with large news article data, and the view needs only the headline, then the presenter can query the data from the model and only give the headline to the view. In this way, the view remains very light in this design pattern.

101. How can we prevent users from creating more than one instance of singleton object by using clone() method?

First we should not implement the Cloneable interface by the object that is a Singleton. Second, if we have to implement Cloneable interface then we can throw exception in clone() method. This will ensure that no one can use clone() method or Cloneable interface to create more than one instance of Singleton object.

102. What is the use of Interceptor design pattern?

Interceptor design pattern is used for intercepting a request. Primary use of this pattern is in Security policy implementation. We can use this pattern to intercept the requests by a client to a resource. At the interception we can check for authentication and authorization of client for the resource being accessed. In Java it is used in javax.servlet.Filter interface. This pattern is also used in Spring framework in HandlerInterceptor and MVC interceptor.

103. What are the Architectural patterns that you have used?

Architectural patterns are used to define the architecture of a Software system. Some of the patterns are as follows:

- **MVC:** Model View Controller. This pattern is extensively used in the architecture of Spring framework.
- **Publish-subscribe:** This pattern is the basis of messaging architecture. In this case messages are published to a Topic. And subscribers subscribe to the topic of their interests. Once the message is published to a topic in which a Subscriber has an interest, the message is consumed by the relevant subscriber.
- **Service Locator:** This design pattern is used in a service like JNDI to locate the available services. It uses as central registry to maintain the list of services.
- **n-Tier:** This is a generic design pattern to divide the architecture in multiple tiers. E.g. there is 3-tier architecture with Presentation layer, Application layer and Data access layer. It is also called multi-layer design pattern.
- **Data Access Object (DAO):** This pattern is used in providing access to database objects. The underlying principle is that we can change the underlying database system, without changing the business logic. Since business logic talks to DAO object, there is no impact of changing Database system on business logic.
- **Inversion of Control (IoC):** This is the core of Dependency Injection in Spring framework. We use this design pattern to increase the modularity of an application. We keep the objects loosely coupled with Dependency Injection.

104. What are the popular uses of Façade design pattern?

Some of the popular uses of Façade design pattern are as follows:

- A Façade provides convenient methods for common tasks that are used more often.
- A Façade can make the software library more readable.
- A Façade can reduce the external dependencies on the working of inner code.
- A Façade can act as a single well-designed API by wrapping a collection of poorly designed APIs.
- A Façade pattern can be used when a System is very complex and difficult to use. It can simplify the usage of complex system.

105. What is the difference between Builder design pattern and Factory design pattern?

Both Factory and Builder patterns are creational design patterns. They are similar in nature but Factory pattern is a simplified generic version of Builder pattern. We use Factory pattern to create different concrete subtypes of an Object.

The client of a Factory may not know the exact subtype. E.g. If we call `createDrink()` of a Factory, we may get Tea or Coffee drinks. We can also use Builder pattern to create different concrete subtypes of an object. But in the Builder pattern the composition of the object can be more complex. E.g. If we call `createDrink()` for Builder, we can get Cappuccino Coffee with Vanilla Cream and Sugar, or we can get Latte Coffee with Splenda and milk cream. So a Builder can support creation of a large number of variants of an object. But a Factory can create a broader range of known subtypes of an object.

106. What is Memento design pattern?

Memento design pattern is used to implement rollback feature in an object. In a Memento pattern there are three objects: Originator: This is the object that has an internal state. Caretaker: This is the object that can change the state of Originator. But it wants to have control over rolling back the change. Memento: This is the object that Caretaker gets from Originator, before making any change. If Caretaker wants to Rollback the change it gives Memento back to Originator. Originator can use Memento to restore its own state to the original state. E.g. One good use of memento is in online Forms. If we want to show to user a form pre-populated with some data, we keep this copy in memento. Now user can update the form. But at any time when user wants to reset the form, we use memento to make the form in its original pre-populated state. If user wants to just save the form we save the form and update the memento. Now onwards any new changes to the form can be rolled back to the last saved Memento object.

107. What is an AntiPattern?

An AntiPattern is opposite of a Design Pattern. It is a common practice in an organization that is used to deal with a recurring problem but it has more bad consequences than good ones. AntiPattern can be found in an Organization, Architecture or Software Engineering. Some of the AntiPatterns in Software Engineering are:

- Gold Plating: Keep on adding extra things on a working solution even though these extra things do not add any additional value.
- Spaghetti Code: Programs that are written in a very complex way and are hard to understand due to misuse of data structures.
- Coding By Exception: Adding new code just to handle exception cases and corner case scenarios.
- Copy Paste Programming: Just copying the same code multiple times rather than writing generic code that can be parameterized.

108. What is a Data Access Object (DAO) design pattern?

DAO design pattern is used in the data persistent layer of a Java application. It mainly uses OOPS principle of Encapsulation. By using DAO pattern it makes the application loosely coupled and less dependent on actual database. We can even implement some in-memory database like H2 with DAO to handle the unit-testing. In short, DAO hides the underlying database implementation from the class that accesses the data via DAO object. Recently we can combine DAO with Spring framework to inject any DB implementation.

109. What is Spring framework?

Spring is development framework for Java programming. It is an open source development framework for Enterprise Java. The core features of Spring Framework can be used in developing a Java Enterprise application. It has many extensions and jars for developing web applications on top of Java EE platform. With Spring we can develop large-scale complex Java applications very easily. It is also based on good design patterns like Dependency Injection, Aspect oriented programming for developing extensible feature rich software.

110. What are the benefits of Spring framework in software development?

Many benefits of Spring framework are:

- Lightweight Framework: Basic Spring framework is very small in size. It is easy to use and does not add a lot of overhead on software. It just has 2 MB in basic version.
- Container: Spring framework provides the basic container that creates and manages the life cycle of application objects like Plain old Java objects (POJO). It also stores the configuration files of application objects to be created.

- **Dependency Injection (DI):** Spring provided loose coupling is application by Dependency Injection. It uses Inversion of Control technique by which objects specify their dependencies to Spring container instead of creating new objects themselves.
- **Aspect Oriented Programming (AOP):** Spring framework promotes and provides support for Aspect oriented programming in Java. This helps in separating application business logic from system services that are common across all the business logic. E.g. Logging can be a cross cutting concern in an Application.
- **Transaction Management:** Spring provides a framework for transaction management. So a developer does not have to implement it from scratch. Spring Transaction Management is so powerful that we can scale it from one local transaction to global transactions in a cluster.
- **MVC Framework:** For Web applications, Spring provides MVC framework. This framework is based on MVC design pattern and has better features compared to other web frameworks.
- **Exception Handling:** Spring also gives support for a common API to handle exceptions in various technologies like Hibernate, JDBC etc.

111. What are the modules in Core Container of Spring framework?

Spring framework has a Core Container. Modules in Core Container are:

- Core module
- Bean module
- Context module
- Spring Expression Language module

112. What are the modules in Data Access/Integration layer of Spring framework?

Modules in Data Access/Integration Layer of Spring framework are:

- **JDBC module:** An abstraction layer to remove tedious JDBC coding.
- **ORM module:** Integration layers for Object Relational Mapping
- **OXM module:** An abstraction layer to support Object XML mapping.
- **Java Messaging Service (JMS) module:** Module for producing and consuming messages.
- **Transactions module:** Transaction Management for POJO classes

113. What are the modules in Web layer of Spring framework?

Modules in Web Layer of Spring framework are:

- **Web module:** This provides basic web-oriented integration features.
- **Servlet module:** Support for Servlet Listeners.
- **WebSocket module:** Support for Web Socket style messaging.
- **Portlet module:** MVC implementation for Portlet environment.

114. What is the main use of Core Container module in Spring framework?

As the name suggests, Spring Core Container is the core of Spring framework. It gives the basic functionality of the Spring. All the parts of Spring Framework are built on top of Core Container. Its main use is to provide Dependency Injection (DI) and Inversion of control (IOC) features.

115. What kind of testing can be done in Spring Test Module?

Spring Test Module provides support for Unit testing as well as Integration testing of Spring components. It allows using JUnit or TestNG testing frameworks. It also gives ability to mock objects to use the test code.

116. What is the use of BeanFactory in Spring framework?

BeanFactory is the main class that helps in implementing Inversion of Control pattern in Spring. It is based on the factory design pattern. It separates the configuration and dependencies of an application from the rest of application code. Implementations of BeanFactory like XmlBeanFactory class are used by applications built with Spring.

117. Which is the most popular implementation of BeanFactory in Spring?

XMLBeanFactory is the most popular implementation of BeanFactory in Spring.

118. What is XMLBeanFactory in Spring framework?

XMLBeanFactory is one of the most useful implementation of BeanFactory in Spring. This factory loads its beans based on the definitions mentioned in an XML file. Spring container reads bean configuration metadata from an XML file and creates a fully configured application with the help of XMLBeanFactory class.

119. What are the uses of AOP module in Spring framework?

AOP module is also known as Aspect Oriented Programming module. Its uses are:

- Development of aspects in a Spring based application
- Provides interoperability between Spring and other AOP frameworks
- Supports metadata programming to Spring

120. What are the benefits of JDBC abstraction layer module in Spring framework?

Spring provides JDBC abstraction layer module. Main benefits of this module are:

- Helps in keeping the database code clean and simple.
- Prevents problems that result from a failure to close database resources.
- Provides a layer of useful exceptions on top of the error messages given by different database servers.
- Based on Spring's AOP module
- Provides transaction management services for objects in a Spring application

121. How does Spring support Object Relational Mapping (ORM) integration?

Spring supports Object Relational Mapping (ORM) by providing ORM Module. This module helps in integrating with popular ORM framework like Hibernate, JDO, and iBATIS SQL Maps etc. Transaction Management module of Spring framework supports all of these ORM frameworks as well as JDBC.

122. How does Web module work in Spring framework?

Spring provides support for developing web application by using Web module. This module is built on application context module that provides context for web-based applications. This module also supports web-oriented integration features like transparently handling multipart requests for uploading files, programmatically binding request parameters to business objects etc. This module also supports integration with popular web frameworks like Jakarta Struts, JSF, and Tapestry etc.

123. What are the main uses of Spring MVC module?

Spring-webmvc module is also known as Web-servlet module. It is based on Web Model View Controller pattern. Main uses of this module are:

- Integration of Spring with other MVC frameworks
- Supports IoC to provide clean separation of controller logic from business objects
- Provides clean separation between domain model code and web forms
- Allows developers to declaratively bind request parameters to business objects

124. What is the purpose of Spring configuration file?

Spring application can be configured by an XML file. This file contains information of classes and how these classes are configured and introduced to each other. Spring IoC container uses some kind of configuration metadata. This configuration metadata represents how an application developer tells the Spring container to instantiate, configure, and assemble the objects in your application. This configuration metadata is stored in Spring configuration file. The other ways of specifying configuration metadata are Java based configuration and Annotation based configuration.

125. What is the purpose of Spring IoC container?

The Spring IoC Container is responsible for: Creating the objects, Configuring the objects, Managing dependency between objects (with dependency injection (DI)), Wiring the objects together, Managing complete lifecycle of objects.

126. What is the main benefit of Inversion of Control (IOC) principle?

Inversion of Control (IOC) principle is the base of Spring framework. It supports dependency injection in an application. With Dependency Injection, a programmer has to write minimal code. It also makes easier to test an application. Most important benefit is that it leads to loose coupling within objects. With loose coupling it is easier to change the application with new requirements.

127. Does IOC containers support Eager Instantiation or Lazy loading of beans?

IOC Container in Spring supports both the approaches. Eager instantiation as well as lazy loading of beans.

128. What are the benefits of ApplicationContext in Spring?

ApplicationContext in Spring provides following benefits:

- Bean factory methods: These are used to access application components
- Load File Resources: It helps in loading file resources in a generic fashion
- Publish Events: It enables publishing events to registered listeners
- Internationalization Support: Ability to resolve messages to support internationalization
- Parent Context: Ability to inherit from a parent context

129. How will you implement ApplicationContext in Spring framework?

ApplicationContext in Spring can be implemented in one of the following three ways:

- FileSystemXmlApplicationContext: If we want to load the definitions of beans from an XML file then FileSystemXmlApplicationContext is used. The full path of XML bean configuration file is provided to the constructor.
- ClassPathXmlApplicationContext: To load the definitions of beans from an XML file in the CLASSPATH, we use ClassPathXmlApplicationContext. It is used for application context embedded in jars.
- WebXmlApplicationContext: To provide configuration for a web application WebXmlApplicationContext is used. While the application is running, it is read only. But it can be reloaded if underlying application supports it.

130. Explain the difference between ApplicationContext and BeanFactory in Spring?

Main differences between ApplicationContext and BeanFactory are:

- Automatic BeanPostProcessor registration: BeanFactory does not support BeanPostProcessor registration. Whereas ApplicationContext support this.
- Automatic BeanFactoryPostProcessor registration: BeanFactory also does not allow Automatic BeanFactoryPostProcessor registration. Whereas ApplicationContext allows this.
- MessageSource access: BeanFactory is not convenient for MessageSource access. ApplicationContext is quite convenient for MessageSource access.
- ApplicationEvent: We cannot publish ApplicationEvent with BeanFactory. But ApplicationContext provides ability to publish ApplicationEvent.

131. Between ApplicationContext and BeanFactory which one is preferable to use in Spring?

Spring documentation recommends using ApplicationContext in almost all the cases. ApplicationContext has all the functionality of BeanFactory.

132. What are the main components of a typical Spring based application?

In a Spring based application, main components are:

- Spring configuration XML file: This is used to configure Spring application
- API Interfaces: Definition of API interfaces for functions provided by application
- Implementation: Application code with implementation of APIs

- Aspects: Spring Aspects implemented by application
- Client: Application at client side that is used for accessing functions

133. Explain Dependency Injection (DI) concept in Spring framework?

Dependency Injection is a software design pattern. It is used to implement Inversion of Control (IOC) in Spring framework. As per this pattern, we do not create objects in an application by calling new. Rather, we describe how an object should be created. In this way creation of an object is not tightly coupled with another object. A container is responsible for creating and wiring the objects. The container can call injecting code and wire the objects as per the configuration at runtime.

134. What are the different roles in Dependency Injection (DI)?

There are four roles in Dependency Injection:

- Service object(s) to be used
- Client object that depends on the service
- Interface that defines how client uses services
- Injector responsible for constructing services and injecting them into client

135. Spring framework provides what kinds of Dependency Injection mechanism?

Spring framework provides two types of Dependency Injection mechanism:

- Constructor-based Dependency Injection: Spring container can invoke a class constructor with a number of arguments. This represents a dependency on other class.
- Setter-based Dependency Injection: Spring container can call setter method on a bean after creating it with a no-argument constructor or no-argument static factory method to instantiate another bean.

136. In Spring framework, which Dependency Injection is better? Constructor-based DI or Setter-based DI?

Spring framework provides support for both Constructor-based and Setter-based Dependency Injection. There are different scenarios in which these options can be used. It is recommended to use Constructor-based DI for mandatory dependencies. Whereas Setter-based DI is used for optional dependencies.

137. What are the advantages of Dependency Injection (DI)?

Dependency Injection (DI) pattern has the following advantages:

- Dependency Injection reduces coupling between a class and its dependencies.
- With Dependency Injection (DI), we can do concurrent or independent software development. Two teams can work in parallel on classes that will be used by each other.
- In Dependency Injection (DI), the client can be configured in multiple ways. It needs to just work with the given interface. The rest of the implementation can be changed and configured for different features.
- Dependency injection is also used to export a system's configuration details into configuration files. So we can configure the same application to run in different environments based on configuration, e.g., Test environment, UAT environment, and Production environment.
- Dependency Injection (DI) applications provide more ease and flexibility of testing. These can be tested in isolation in Unit Test.
- Dependency injection (DI) isolates the client from the impact of design and implementation changes. Therefore, it promotes reusability, testability, and maintainability.

138. What are the disadvantages of Dependency Injection (DI)?

Dependency Injection (DI) pattern has the following disadvantages:

- Most of the time Dependency Injection forces developers to use an injection framework like Spring. This causes dependency on a framework.
- With Dependency Injection, clients are dependent on the configuration data. This becomes an extra task for developers when the application does not need so many custom configuration values.

- Code is difficult to trace and read in Dependency Injection. DI separates behavior from construction of objects.
- Dependency injection increases complexity in the linkages between classes. It may become harder to manage such complexity outside the implementation of a class.

139. What is a Spring Bean?

A Spring Bean is a plain old Java object (POJO) that is created and managed by a Spring container. There can be more than one bean in a Spring application, but all these beans are instantiated and assembled by the Spring container. The developer provides configuration metadata to the Spring container for creating and managing the lifecycle of a Spring Bean. In general, a Spring Bean is singleton. Every bean has an attribute named "singleton." If its value is true then the bean is a singleton. If its value is false then the bean is a prototype bean. By default, the value of this attribute is true. Therefore, by default all the beans in the Spring framework are singleton in nature.

140. What does the definition of a Spring Bean contain?

A Spring Bean definition contains configuration metadata for the bean. This configuration metadata is used by the Spring container to:

- Create the bean
- Manage its lifecycle
- Resolve its dependencies

141. What are the different ways to provide configuration metadata to a Spring Container?

Spring supports three ways to provide configuration metadata to Spring Container: XML based configuration: We can specify configuration data in an XML file. Annotation-based configuration: We can use Annotations to specify configuration. This was introduced in Spring 2.5. Java-based configuration: This is introduced from Spring 3.0. We can embed annotations like @Bean, @Import, @Configuration in Java code to specify configuration metadata.

142. What are the different scopes of a Bean supported by Spring?

Spring framework supports seven types of scopes for a Bean. Out of these only five scopes are available for a web-aware ApplicationContext application:

- singleton: This is the default scope of a bean. Under this scope, there is a single object instance of bean per Spring IoC container.
- prototype: Under this scope a single bean definition can have multiple object instances.
- request: In this scope, a single bean definition remains tied to the lifecycle of a single HTTP request. Each HTTP request will have its own instance of a bean for a single bean definition. It is only valid in the context of a web-aware Spring ApplicationContext.
- session: Under this scope, a single bean definition is tied to the lifecycle of an HTTP Session. Each HTTP Session will have one instance of bean. It is also valid in the context of a web-aware Spring ApplicationContext.
- globalSession: This scope ties a single bean definition to the lifecycle of a global HTTP Session. It is generally valid in a Portlet context. It is also valid in the context of a web-aware Spring ApplicationContext.
- application: This scope limits a single bean definition to the lifecycle of a ServletContext. It is also valid in the context of a web-aware Spring ApplicationContext.
- websocket: In this scope, a single bean definition is tied to the lifecycle of a WebSocket. It is also valid in the context of a web-aware Spring ApplicationContext.

143. How will you define the scope of a bean in Spring?

In configuration xml, we can specify the scope of bean in its definition. This is used by container to decide the scope of bean in Spring.

Example:

XML

```
<bean id="userService" class="com.um.UserService" scope="prototype"/>
```

Show more lines

This is an example of userService bean with prototype scope.

144. Is it safe to assume that a Singleton bean is thread safe in Spring Framework?

No, Spring framework does not guarantee anything related to multi-threaded behavior of a singleton bean. Developer is responsible for dealing with concurrency issues and maintaining thread safety of a singleton bean.

145. What are the design-patterns used in Spring framework?

Spring framework uses many Design patterns. Some of these patterns are:

- Singleton – By default beans defined in spring config files are singleton. These are based on Singleton pattern.
- Template – This pattern is used in many classes like JdbcTemplate, RestTemplate, JmsTemplate, JpaTemplate etc.
- Dependency Injection – This pattern is the core behind the design of BeanFactory and ApplicationContext.
- Proxy – Aspect Oriented Programming (AOP) heavily uses proxy design pattern.
- Front Controller – DispatcherServlet in Spring is based on Front Controller pattern to ensure that incoming requests are dispatched to other controllers.
- Factory pattern – To create an instance of an object, BeanFactory is used. This is based on Factory pattern.
- View Helper – Spring has multiple options to separating core code from presentation in views. Like Custom JSP tags, Velocity macros etc.

146. What is the lifecycle of a Bean in Spring framework?

A Bean in Spring framework goes through following phases in its lifecycle. Initialization and creation: Spring container gets the definition of Bean from XML file and instantiates the Bean. It populates all the properties of Bean as mentioned in the bean definition. Setting the Behavior of Bean: In case a Bean implements BeanNameAware interface, Spring uses setBeanName() method to pass the bean's id. In case a Bean implements BeanFactoryAware interface, Spring uses setBeanFactory() to pass the BeanFactory to bean. Post Processing: Spring container uses postProcessorBeforeInitialization() method to call BeanPostProcessors associated with the bean. Spring calls afterPropertySet() method to call the specific initialization methods. In case there are any BeanPostProcessors of a bean, the postProcessAfterInitialization() method is called. Destruction: During the destruction of a bean, if bean implements DisposableBean, Spring calls destroy() method.

147. What are the two main groups of methods in a Bean's lifecycle?

A Bean in Spring has two main groups of lifecycle methods.

- Initialization Callbacks: Once all the necessary properties of a Bean are set by the container, Initialization Callback methods are used for performing initialization work. A developer can implement method afterPropertiesSet() for this work.
- Destruction Callbacks: When the Container of a Bean is destroyed, it calls the methods in DisposableBean to do any cleanup work. There is a method called destroy() that can be used for this purpose to make Destruction Callbacks.

Recent recommendation from Spring is to not use these methods, since it can strongly couple your code to Spring code.

148. Can we override main lifecycle methods of a Bean in Spring?

Yes, Spring framework allows developers to override the lifecycle methods of a Bean. This is used for writing any custom behaviour for Bean.

149. What are Inner beans in Spring?

A bean that is used as a property of another bean is known as Inner bean. It can be defined as a element in or tags. It is not mandatory for an Inner bean to have id or a name. These are always anonymous. Inner bean does not need a scope. By default it is of prototype scope.

150. How can we inject a Java Collection in Spring framework?

Spring promotes Dependency Injection (DI) in code. It gives support for injecting not only objects but also collection of objects. We can inject collections like list, set, map etc. in Spring. Following tags can be used for this purpose:

- `<list>`: This type is used for injecting a list of values. In a duplicates are allowed.
- `<set>`: This type is used for injecting a set of values. As per set property, duplicates are not allowed.
- `<map>`: This type is used for injecting name-value pairs in form of map. Name and value can be of any type that is allowed for a map.
- `<properties>`: This type is used to inject a collection of String based name-value. It is like a properties file.

151. What is Bean wiring in Spring?

A Spring container is responsible for injecting dependencies between beans. This process of connecting beans is called wiring. Developer mentions in configuration file, the dependencies between beans. And Spring container reads these dependencies and wires the beans on creation.

152. What is Autowiring in Spring?

Autowiring is a feature of Spring in which container can automatically wire/connect the beans by reading the configuration file. Developer has to just define “autowire” attribute in a bean. Spring resolves the dependencies automatically by looking at this attribute of beans that are autowired.

153. What are the different modes of Autowiring supported by Spring?

There are five modes of Autowiring supported by Spring framework:

- `no`: This is default setting for Autowiring. In this case, we use “ref” mode to mention the explicit bean that is being referred for wiring. Example:

XML

```
<bean id="employee" class="com.dept.Employee">
<property name="manager" ref="manager" />
</bean>
<bean id="manager" class="com.dept.Manager" />
```

Show more lines

- `byName`: In this case, Spring container tries to match beans by name during Autowiring. If the name of a bean is same as the name of bean referred in autowire byname, then it automatically wires it. Example:

XML

```
<bean id="employee" class="com.dept.Employee" autowire="byName" />
<bean id="manager" class="com.dept.Manager" />
```

Show more lines

- `byType`: In this case, Spring container check the properties of beans referred with attribute byType. Then it matches the type of bean and wires. If it finds more than one such bean of that type, it throws a fatal exception. Example:

XML

```
<bean id="employee" class="com.dept.Employee" autowire="byType" />
<bean id="manager" class="com.dept.Manager" />
```

Show more lines

- constructor: In this case, Spring container looks for byType attribute in constructor argument. It tries to find the bean with exact name. If it finds more than one bean of same name, it throws fatal exception. This case is similar to byType case. Example:

XML

```
<bean id="employee" class="com.dept.Employee" autowire="constructor" />
<bean id="manager" class="com.dept.Manager" />
```

Show more lines

- autodetect: This is an advanced mode for autowiring. In this case, by default Spring tries to find a constructor match. If it does not find constructor then it uses autowire by Type. Example:

XML

```
<bean id="employee" class="com.dept.Employee" autowire="autodetect" />
<bean id="manager" class="com.dept.Manager" />
```

Show more lines

154. What are the cases in which Autowiring may not work in Spring framework?

Autowiring is a great feature in Spring. It can be used in most of the cases. But there are certain scenarios in which Autowiring may not work.

- Explicit wiring: Since Autowiring is done by Spring, developer does not have full control on specifying the exact class to be used. It is preferable to use Explicit wiring in case of full control over wiring.
- Primitive Data types: Autowiring does not allow wiring of properties that are based on primitive data types like int, float etc.

155. Is it allowed to inject null or empty String values in Spring?

Yes, Spring allows injecting null or empty String values.

156. What is a Java-based Configuration in Spring?

Spring allows for Java-based configuration in which a developer can specify configuration by using Java-based annotations. This feature was introduced in Spring 3.0. You can use annotations like @Configuration, @Bean, @Import and @DependsOn in Java classes for specifying the configuration.

157. What is the purpose of @Configuration annotation?

This annotation is used in a class to indicate that this is class is the primary source of bean definitions. This class can also contain inter-bean dependencies that are annotated by @Bean annotation.

158. What is the difference between Full @Configuration and 'lite' @Beans mode?

Spring allows for using @Bean annotation on methods that are declared in classes not annotated with @Configuration. This is known as "lite" mode. In this mode, bean methods can be declared in a @Component or a plain java class without any annotation. In the "lite" mode, @Bean methods cannot declare inter-bean dependencies. It is recommended that one @Bean method should not invoke another @Bean method in 'lite' mode. Spring recommends that @Bean methods declared within @Configuration classes should be used for full configuration. This kind of full mode can prevent many bugs.

159. In Spring framework, what is Annotation-based container configuration?

From Spring 2.5 version it is possible to provide configuration by using annotation. To turn this configuration on, we need to mention <context:annotation-config/> in spring XML file. Now developer can use annotations like @Required, @Autowired, @Qualifier etc. in a class file to specify the configuration for beans. Spring container can use this information from annotation for creating and wiring the beans.

160. How will you switch on Annotation based wiring in Spring?

To use Annotation based wiring, we need to turn on Annotation based configuration in Spring. By default, Annotation based configuration is switched off in Spring. To turn it is we can specify <context:annotation-config/> element in Spring

config file. Once it is turned on, we can use `@Autowired` annotation or `@Required` annotation in a Java class for wiring in Spring.

161. What is `@Autowired` annotation?

We can use `@Autowired` annotation to auto wire a bean on a setter method, constructor or a field. `@Autowired` auto wiring is done by matching the data type. Before using `@Autowired` annotation we have to register `AutowiredAnnotationBeanPostProcessor`. This can be done by including `<context:annotation-config />` in bean configuration file.

162. What is `@Required` annotation?

We use `@Required` annotation to a property to check whether the property has been set or not. Spring container throws `BeanInitializationException` if the `@Required` annotated property is not set. When we use `@Required` annotation, we have to register `RequiredAnnotationBeanPostProcessor` in Spring config file.

163. What are the two ways to enable `RequiredAnnotationBeanPostProcessor` in Spring?

`RequiredAnnotationBeanPostProcessor` can be enabled in two ways in Spring:

- Include `<context:annotation-config />` Add Spring context and `<context:annotation-config />` in bean configuration file. Example:

XML

```
<beans
...
xmlns:context="http://www.springframework.org/schema/context"
...
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd">
...
<context:annotation-config />
...
</beans>
```

Show more lines

- Include `RequiredAnnotationBeanPostProcessor` in bean configuration file Example:

XML

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/>
<bean id="BookBean" class="com.foo.Book">
<property name="action" value="price" />
<property name="type" value="1" />
</bean>
<bean id="AuthorBean" class="com.foo.Author">
<property name="name" value="Rowling" />
</bean>
</beans>
```

Show more lines

164. What is @Qualifier annotation in Spring?

We use @Qualifier annotation to mark a bean as ready for auto wiring. This annotation is used along with @Autowired annotation to specify the exact bean for auto wiring by Spring container.

165. How does Spring framework make JDBC coding easier for developers?

Spring provides a mature JDBC framework to provide support for JDBC coding. Spring JDBC handles resource management as well as error handling in a generic way. This reduces the work of software developers. They just have to write queries and related statements to fetch the data or to store the data in database.

166. What is the purpose of JdbcTemplate?

Spring framework provides JdbcTemplate class that contains many convenient methods for regular tasks like converting data into primitives or objects, executing prepared or callable statements etc. This class makes it very easy to work with database in our Application and it also provides good support for custom error handling in database access code.

167. What are the benefits of using Spring DAO?

Some of the benefits of using Spring DAO are:

- It makes it easier to work on different data access methods like JDBC, Hibernate etc.
- It provides a consistent and common way to deal with different data access methods.
- Spring DAO makes it easier to switch between different data persistence frameworks.
- No need for catching framework specific exceptions.

168. What are the different ways to use Hibernate in Spring?

Spring provides two ways to use Hibernate:

- We can extend HibernateDAOSupport and apply an AOP interceptor node to use Hibernate.
- We can also use HibernateTemplate and Callback to access Hibernate. This is based on Inversion of Control.

169. What types of Object Relational Mapping (ORM) are supported by Spring?

Spring supports following Object Relational Mapping (ORM) frameworks:

- Hibernate
- Java Persistence API (JPA)
- TopLink
- Java Data Objects (JDO)
- Apache Object Relational Bridge (ORB)

170. How will you integrate Spring and Hibernate by using HibernateDaoSupport?

We can use following steps for integrating Spring and Hibernate:

- Add dependencies for Spring and Hibernate in pom.xml
- Implement DAO from HibernateDaoSupport
- Use Hibernate functions via getHibernateTemplate() method

171. What are the different types of the Transaction Management supported by Spring framework?

Spring framework provides support for two types of Transaction Management:

- Programmatic: In this method, we have to manage Transaction by programming explicitly. It provides flexibility to a developer, but it is not easier to maintain.
- Declarative: In this approach, we can separate Transaction Management from the Application Business code. We can use annotations or XML based configuration to manage the transactions in declarative approach.

172. What are the benefits provided by Spring Framework's Transaction Management?

Main benefits provided by Spring Transaction Management are:

- Consistent: By using Spring Transaction management, we can use consistent programming model across different transaction APIs like JPA, JDBC, JTA, Hibernate, JPA, JDO etc.
- Simplicity: Spring TM provides simple API for managing the transaction programmatically.
- Declarative: Spring also supports annotation or xml based declarative transaction management.
- Integration: Spring Transaction management is easier to integrate with other data access abstractions of Spring.

173. Given a choice between declarative and programmatic Transaction Management, which method will you choose?

In Spring, Declarative Transaction Management is the preferred choice. This method is very less invasive and it has very less impact in Application Business Logic. Although Declarative method gives less flexibility than Programmatic method, it is simpler to use and easier to maintain in long run.

174. What is Aspect Oriented Programming (AOP)?

Aspect Oriented Programming (AOP) is a programming paradigm that promotes programmers to develop code in different modules that can be parallel or in crosscutting concerns. E.g. To develop banking software, one team can work on business logic for Money withdrawal, Money deposit, Money Transfer etc. The other team can work on Transaction Management for committing the transaction across multiple accounts.

In an Auto company, one team can work on software to integrate with different components of car. The other team can work on how all the components will send signal and current information to a common dashboard.

175. What is an Aspect in Spring?

An Aspect is the core construct of AOP. It encapsulates the behavior that affects multiple classes in a reusable module. An Aspect can have a group of APIs that provide cross-cutting features. E.g. A logging module can be an Aspect in an Application. An application can have multiple Aspects based on the different requirements. An Aspect can be implemented by using annotation @Aspect on a class.

176. In Spring AOP, what is the main difference between a Concern and a Cross cutting concern?

A Concern in Spring is the behavior or expectation from an application. It can be the main feature that we want to implement in the application. A Cross cutting concern is also a type of Concern. It is the feature or functionality that is spread throughout the application in a thin way. E.g. Security, Logging, Transaction Management etc. are cross cutting concerns in an application.

177. What is a Joinpoint in Spring AOP?

In Spring AOP, Joinpoint refers to a candidate point in application where we can plug in an Aspect. Joinpoint can be a method or an exception or a field getting modified. This is the place where the code of an Aspect is inserted to add new behavior in the existing execution flow.

178. What is an Advice in Spring AOP?

An Advice in Spring AOP is an object containing the actual action that an Aspect introduces. An Advice is the code of cross cutting concern that gets executed. There are multiple types of Advice in Spring AOP.

179. What are the different types of Advice in Spring AOP?

Spring AOP provides five kinds of Advice:

- Before Advice: This type of advice runs just before a method executes. We can use @Before annotation for this.
- After (finally) Advice: This type of advice runs just after a method executes. Even if the method fails, this advice will run. We can use @After annotation here.
- After Returning Advice: This type of advice runs after a method executes successfully. @AfterReturning annotation can be used here.
- After Throwing Advice: This type of advice runs after a method executes and throws an exception. The annotation to be used is @AfterThrowing.

- **Around Advice:** This type of advice runs before and after the method is invoked. We use `@Around` annotation for this.

180. What is a Pointcut in Spring AOP?

A Pointcut in Spring AOP refers to the group of one or more Joinpoints where an advice can be applied. We can apply Advice to any Joinpoint. But we want to limit the places where a specific type of Advice should be applied. To achieve this we use Pointcut. We can use class names, method names or regular expressions to specify the Pointcuts for an Advice.

181. What is an Introduction in Spring AOP?

In Spring AOP we can declare additional methods or fields on behalf of a type. To do this we use an Introduction. It is also known as inter-type declaration. E.g. We can use an Introduction for making a bean implement `IsModified` interface.

182. What is a Target object in Spring AOP?

A Target object is the object that gets Advice from one or more Aspects. This is also known as advised object. In most cases it is a proxy object.

183. What is a Proxy in Spring AOP?

In Spring AOP, a Proxy is an object created by the AOP framework to implement Aspect contracts. It is generally a JDK dynamic proxy or CGLIB proxy.

184. What are the different types of AutoProxy creators in Spring?

Spring AOP provides following standard types of Autoproxy creators:

- **BeanNameAutoProxyCreator:** This is a `BeanPostProcessor` that creates AOP proxies for beans automatically by matching names.
- **DefaultAdvisorAutoProxyCreator:** This creator is more powerful than other Proxy Creators. This also applies eligible advisors automatically to bean in the current context.
- **AbstractAdvisorAutoProxyCreator:** This is the parent class of `DefaultAdvisorAutoProxyCreator`. We can create our own auto-proxy creators by extending this class.

185. What is Weaving in Spring AOP?

In Aspect oriented programming, linking Aspects with the other application types creates an Advised object. This process is known as Weaving. Without Weaving, we just have definition of Aspects. Weaving makes us realize full potential of the AOP. Weaving can be done at compile time, load time or at run time.

186. In Spring AOP, Weaving is done at compile time or run time?

Spring container performs Weaving at run time.

187. What is XML Schema-based Aspect implementation?

Spring allows for implementing Aspect by using regular classes and XML based configurations. This is different from Annotation based Aspect implementation. But it achieves the same goal of AOP. We can use elements like `<aop:aspect id="testAspect" ref="testBean" />` and `<aop:pointcut id="testPointcut" />` in Spring XML config file. To use this we need to import Spring AOP schema as follows:

XML

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/bean
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
```

Show more lines

188. What is Annotation-based aspect implementation in Spring AOP?

This is a declarative style AOP implementation. In this case, we use annotations like @Aspect, @Pointcut, @Joinpoint etc. to annotate code with different types of AOP elements. This can be used Java 5 onwards, when the support for Annotations was introduced.

189. How does Spring MVC framework work?

Spring provides its own Model View Controller (MVC) framework for developing web applications. Spring MVC framework is based on Inversion of Control (IOC) principle. It separates the business objects from controller. It is designed around the DispatcherServlet that is responsible for dispatching requests to relevant handlers. Spring MVC framework also supports annotation based binding of request parameters.

190. What is DispatcherServlet?

In Spring MVC, DispatcherServlet is the core servlet that is responsible for handling all the requests and dispatching these to handlers. Dispatcher servlet knows the mapping between the method to be called and the browser request. It calls the specific method and combines the results with the matching JSP to create an html document, and then sends it back to browser. In case of RMI invocation, it sends back response to the client application.

191. Can we have more than one DispatcherServlet in Spring MVC?

Yes, a Spring MVC web application can have more than one DispatcherServlets. Each DispatcherServlet has to operate in its own namespace. It has to load its own ApplicationContext with mappings, handlers, etc. Only the root application context will be shared among these Servlets.

192. What is WebApplicationContext in Spring MVC?

WebApplicationContext is the child of plain ApplicationContext. It is used in web applications. It provides features to deal with web-related components like controllers, view resolvers etc. A Web Application can have multiple WebApplicationContext to handle requests. Each DispatcherServlet is associated with one WebApplicationContext.

193. What is Controller in Spring MVC framework?

Controller is an interface in Spring MVC. It receives HttpServletRequest and HttpServletResponse in web app just like an HttpServlet, but it is able to participate in an MVC flow. Controllers are similar to a Struts Action in a Struts based Web application. Spring recommends that the implementation of Controller interface should be a reusable, thread-safe class, capable of handling multiple HTTP requests throughout the lifecycle of an application. It is preferable to implement Controller by using a JavaBean. Controller interprets user input and transforms it into a model. The model is represented to the user by a view. Spring implements a controller in a very generic way. This enables us to create a wide variety of controllers.

What is @Controller annotation in Spring MVC?

We use @Controller annotation to indicate that a class is a Controller in Spring MVC. The dispatcher in Spring scans for @Controller annotated classes for mapped methods and detects @RequestMapping.

194. What is @RequestMapping annotation in Spring?

In Spring MVC, we use @RequestMapping annotation to map a web request to either a class or a handler method. In @RequestMapping we can specify the path of URL as well as HTTP method like GET, PUT, POST etc. @RequestMapping also supports specifying HTTP Headers as attributes. We can also map different media types produced by a controller in @RequestMapping. We use HTTP Header Accepts for this purpose. E.g.

Java

```
@RequestMapping(  
value = "/test/mapping",  
method = GET,
```



```
headers = "Accept=application/json"
```

```
)
```

Show more lines

195. What are the main features of Spring MVC?

Spring MVC has following main features:

- Clear separation of role: In Spring MVC, each role like controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver etc. is fulfilled by a specialized object.
- Reusability: Spring MVC promotes reusable business code that reduces the need for duplication. We can use existing business objects as command or form objects instead of copying them to extend a particular framework base class.
- Flexible Model Transfer: Spring MVC Model transfer supports easy integration with other view technologies as well.
- Customizable binding and validation: In Spring MVC, we can do custom binding between Requests and Controllers. Even validation can be done on non-String values as well.
- JSP form tag library: From Spring 2.0, there is a powerful JSP form tag library that makes writing forms in JSP pages much easier.
- Customizable locale, time zone and theme resolution: Spring MVC supports customization in locale, timezone etc.

196. What is the difference between a Singleton and Prototype bean in Spring?

Every bean in Spring has a scope that defines its existence timeframe in the application. Singleton scope for bean limits a bean to a single object instance per Spring IOC container. This single instance is limited to a specific ApplicationContext. If there are multiple ApplicationContext then we can have more than one instance of bean. By default all the beans in Spring framework are Singleton scope beans. With Prototype scope a single bean definition can have multiple object instances in a Spring container. In prototype scope bean, the Spring IoC container creates new bean instance of the object every time a request for that specific bean is made.

197. How will you decide which scope—Prototype or Singleton—to use for a bean in Spring?

In general, we use prototype scope for all stateful beans and singleton scope for stateless beans. Since a stateless bean does not maintain any state, we can use the same object instance again and again. Singleton scope bean serves the same purpose. In a stateful bean, there is a need to maintain the state in each request, it is necessary to use a new instance of object with each call. A Prototype scope bean ensures that we get a new instance each time we request for the object.

198. What is the difference between Setter and Constructor based Dependency Injection (DI) in Spring framework?

Main differences between Setter and Constructor based Dependency Injection (DI) in Spring are:

- Priority: Setter based injection has higher priority than a constructor based injection in Spring. If an application uses Setter as well as Constructor injection, Spring container uses the Setter injection.
- Partial dependency: We can inject partial dependency by using Setter injection. In Constructor injection, it is not possible to do just a partial dependency injection. E.g. If there are two properties in a class, we can use Setter method to inject just one property in the class.
- Flexibility: Setter injection gives more flexibility in introducing changes. One can easily change the value by Setter injection. In case of Constructor injection a new bean instance has to be created always.
- Readability: Setter injection is more readable than Constructor injection. Generally Setter method name is similar to dependency class being used in setter method.

199. What are the drawbacks of Setter based Dependency Injection (DI) in Spring?

Although Setter based Dependency Injection has higher priority than Constructor based DI, there are some disadvantages of it.

- **No Guarantee:** In Setter based DI, there is no guarantee that a certain dependency is injected or not. We may have an object with partial or no dependency. Whereas in Constructor based DI, an object is not created till the time all the dependencies are ready.
- **Security:** One can use Setter based DI to override another dependency. This can cause Security breach in a Spring application.
- **Circular Dependency:** Setter based DI can cause circular dependency between objects. Whereas Constructor based DI will throw `ObjectCurrentlyInCreationException` if there is a circular dependency during the creation of an object.

200. What are the differences between Dependency Injection (DI) and Factory Pattern?

Main differences between Dependency Injection (DI) and Factory Pattern are:

- **Coupling:** Factory pattern adds tight coupling between an object, factory and dependency. In case of DI, there is no coupling between objects. We just mention the dependencies on different objects and container resolves and introduces these dependencies.
- **Easier Testing:** DI is easier to test, since we can inject the mock objects as dependency in Test environment. In case of Factory pattern, we need to create actual objects for testing.
- **Flexibility:** DI allows for switching between different DI frameworks easily. It gives flexibility in the choice of DI framework.
- **Container:** DI always needs a container for injecting the dependencies. This leads to extra overhead as well as extra code in your application. In factory pattern, you can just use POJO classes to implement the application without any container.
- **Cleaner Code:** DI code is much cleaner than Factory pattern based code. In DI, we do not need to add extra code for factory methods.

201. In Spring framework, what is the difference between FileSystemResource and ClassPathResource?

In Spring we can specify configuration by using a file or classpath. In `FileSystemResource` we have to give absolute path or relative path of Spring Configuration file (`spring-config.xml` file). In `ClassPathResource`, Spring looks for the Spring Configuration file (`spring-config.xml`) in the ClassPath. Therefore, the developer has to include `spring-config.xml` in the classpath. `ClassPathResource` looks for the configuration file in `CLASSPATH`, whereas `FileSystemResource` looks for the configuration file in the file system.

202. Name some popular Spring framework annotations that you use in your project?

Spring has many annotations to serve different purposes. For regular use, we refer to the following popular Spring annotations:

- **@Controller:** This annotation is for creating controller classes in a Spring MVC project.
- **@RequestMapping:** This annotation maps the URI to a controller handler method in Spring MVC.
- **@ResponseBody:** For sending an Object as response we use this annotation.
- **@PathVariable:** To map dynamic values from a URI to handler method arguments, we use this annotation.
- **@Autowired:** This annotation indicates to Spring for auto-wiring dependencies in beans.
- **@Service:** This annotation marks the service classes in Spring.
- **@Scope:** We can define the scope of Spring bean by this annotation.
- **@Configuration:** This is an annotation for Java-based Spring configuration.
- **@Aspect, @Before, @After, @Around, @Joinpoint, @Pointcut:** These are the annotations in Spring for AspectJ AOP.

203. How can you upload a file in Spring MVC Application?

In Spring MVC framework, we can use the MultipartResolver interface to upload a file. We need to make configuration changes to make it work. After uploading the file, we have to create a Controller handler method to process the uploaded file in the application.

204. What are the different types of events provided by Spring framework?

Spring framework provides the following five events for Context:

- **ContextRefreshedEvent:** Whenever ApplicationContext is initialized or refreshed, Spring publishes this event. We can also raise it by using the refresh() method on ConfigurableApplicationContext interface.
- **ContextStartedEvent:** When ApplicationContext is started using the start() method on ConfigurableApplicationContext interface, ContextStartedEvent is published. We can poll the database or restart any stopped application after receiving this event.
- **ContextStoppedEvent:** Spring publishes this event when ApplicationContext is stopped using the stop() method on ConfigurableApplicationContext interface. This is used for doing any cleanup work.
- **ContextClosedEvent:** Once the ApplicationContext is closed using the close() method, ContextClosedEvent is published. Once a context is closed, it is the last stage of its lifecycle. After this, it cannot be refreshed or restarted.
- **RequestHandledEvent:** This is a web-specific event that informs all beans that an HTTP request has been serviced.

205. What is the difference between DispatcherServlet and ContextLoaderListener in Spring?

DispatcherServlet is the core of a Spring MVC application. It loads the Spring bean configuration file and initializes all the beans mentioned in the config file. In case we have enabled annotations in the Spring config file, it also scans the packages and configures any bean annotated with @Component, @Controller, @Repository, or @Service annotations.

ContextLoaderListener is a listener to start up and shut down Spring's root WebApplicationContext. ContextLoaderListener links the lifecycle of ApplicationContext to the lifecycle of the ServletContext. It automates the creation of ApplicationContext. It can also be used to define shared beans used across different Spring contexts.

206. How will you handle exceptions in Spring MVC Framework?

Spring MVC Framework provides the following mechanisms to help us achieve exception handling:

- **Controller Based:** A developer can define exception handler methods in a Controller class. To do so, they have to annotate the methods with @ExceptionHandler annotation.
- **Global Exception Handler:** Spring provides @ControllerAdvice annotation for exception handling as a cross-cutting concern. We can mark any class as a global exception handler by using this annotation.
- **HandlerExceptionResolver implementation:** Spring Framework provides HandlerExceptionResolver interface that can be implemented to create a global exception handler.

207. What are the best practices of Spring Framework?

In Spring Framework, the following are some of the best practices:

- We can divide Spring bean configurations based on their concerns such as spring-jdbc.xml, spring-security.xml.
- It is better to avoid version numbers in schema reference. This makes sure that we have the latest config files.
- It is a good practice to configure bean dependencies as much as possible. Unless there is a good reason, we try to avoid autowiring.
- For Spring beans that are used in multiple contexts in Spring MVC, we can create them in root context and initialize with listener.
- Spring framework provides many features and modules. We should just use what we need for our application. An extra dependency has to be removed.
- For application properties, it is good to create a property file and read it in Spring configuration file.
- Annotations are useful for smaller applications, but for larger applications annotations can become an overhead. It is easier to maintain if all the configurations are in XML files.

- When we are doing AOP, we have to make sure to keep the Joinpoint as narrow as possible to avoid Advice on unwanted methods.
- We should use the right annotation for components or services. For services use `@Service` and for DAO beans use `@Repository`.
- Dependency Injection (DI) has to be used when there is real benefit. It should not be used just for the sake of loose coupling.

208. What is Spring Boot?

Spring Boot is a ready-made solution to create Spring applications with production-grade features. It favors convention over configuration. We can embed Tomcat or Jetty in an application created with Spring Boot. Spring Boot automatically configures Spring in an application. It does not require any code generation or XML configuration. It is an easy solution to create applications that can run stand-alone.

209. What is Hibernate framework?

Hibernate is a popular Object Relational Mapping (ORM) framework of Java. It helps in mapping the Object Oriented Domain model to Relational Database tables. Hibernate is free software distributed under GNU license. Hibernate also provides implementation of Java Persistence API (JPA). In simple words, it is a framework to retrieve and store data from database tables from Java.

210. What is an Object Relational Mapping (ORM)?

Object Relational Mapping (ORM) is a programming technique to map data from a relational database to an Object Oriented domain model. This is the core of Hibernate framework. In case of Java, most of the software is based on OOPS design. But the data stored in Database is based on Relational Database Management System (RDBMS). ORM helps in data retrieval in an Object Oriented way from an RDBMS. It reduces the effort of developers in writing queries to access and insert data.

211. What is the purpose of Configuration Interface in Hibernate?

Configuration interface can be implemented in an application to specify the properties and mapping documents for creating a SessionFactory in Hibernate. By default, a new instance of Configuration uses properties mentioned in hibernate.properties file. Configuration is mainly an initialization time object that loads the properties and helps in creating SessionFactory with these properties. In short, Configuration interface is used for configuring Hibernate framework in an application.

212. What is Object Relational Impedance Mismatch?

Object Relational Impedance Mismatch (ORIM) is also known as paradigm mismatch. It means that Object model and Relational model do not work well with each other. Relational model or a RDBMS represents data in tabular format like a spreadsheet. Object model or OOPS represents the data as an inter-connected graph of objects. Mixing these two models leads to various problems. The common name for these issues is Object Relational Impedance Mismatch.

213. What are the main problems of Object Relational Impedance Mismatch?

Object model and Relational models (RDBMS) have the following problems that are part of Object Relational Impedance Mismatch:

- Granularity: Object model is more granular than Relational model. There are more classes in object model than the corresponding tables in relational model.
- Inheritance: Object model supports inheritance. But Relational model does not have any concept of inheritance.
- Identity: Relational model has just one criteria for sameness of data. It is based on primary key. In object model like Java we can have equals as well as == for sameness of objects.
- Associations: In Object model associations are uni-directional. In RDBMS, there is a concept of foreign key for association. Also, multiplicity of a relationship is hard to judge by looking at object model.

- Data navigation: In Object model, you can move from one object to another object for getting data. E.g., you can retrieve an Employee object, then go to its department object and then get the employees in the department object. In RDBMS, we try to minimize the SQL calls, so we get all the data by using joins.

214. What are the key characteristics of Hibernate?

Hibernate has the following key characteristics:

- Object/Relational Mapping (ORM): Hibernate provides ORM capabilities to developers. So they can write code in Object model for connecting with data in Relational model.
- JPA Provider: Hibernate provides an excellent implementation of Java Persistence API (JPA) specification.
- Idiomatic persistence: Hibernate provides persistence based on natural Object-oriented idioms with full support for inheritance, polymorphism, association, composition, and the Java collections framework. It can work with any data for persistence.
- High Performance: Hibernate provides high level of performance supporting features like lazy initialization, multiple fetching strategies, optimistic locking, etc. Hibernate does not need its own database tables or fields. It can generate SQL at system initialization to provide better performance at runtime.
- Scalability: Hibernate works well in multi-server clusters. It has built-in scalability support. It can work well for small projects as well as for large business software.
- Reliable: Hibernate is a very reliable and stable framework. This is the reason for its worldwide acceptance and popularity among developer community.
- Extensible: Hibernate is quite generic in nature. It can be configured and extended as per the use case of application.

215. Can you tell us about the core interfaces of Hibernate framework?

The core interfaces of Hibernate framework are as follows:

- Configuration: Configuration interface can be implemented in an application to specify the properties and mapping documents for creating a SessionFactory in Hibernate. Hibernate application bootstraps by using this interface.
- SessionFactory: In Hibernate, SessionFactory is used to create and manage Sessions. Generally, there is one SessionFactory created for one database. It is a thread-safe interface that works well in multi-threaded applications.
- Session: Session is a lightweight object that is used at runtime between a Java application and Hibernate. It contains methods to create, read and delete operations for entity classes. It is a basic class that abstracts the concept of persistence.
- Transaction: This is an optional interface. It is a short-lived object that is used for encapsulating the overall work based on unit of work design pattern. A Session can have multiple Transactions.
- Query: This interface encapsulates the behavior of an object-oriented query in Hibernate. It can accept parameters and execute the queries to fetch results. Same query can be executed multiple times.
- Criteria: This is a simplified API to retrieve objects by creating Criterion objects. It is very easy to use for creating Search-like features.

216. How will you map the columns of a DB table to the properties of a Java class in Hibernate?

We can map the class properties and table columns by using one of the two ways:

- XML: We can map the column of a table to the property of a class in an XML file. It is generally with extension hbm.xml.
- Annotation: We can also use annotations @Entity and @Table to map a column to the property of a class.

217. Does Hibernate make it mandatory for a mapping file to have .hbm.xml extension?

No. It is a convention to have .hbm.xml extension in the name of a mapping file. It is not a requirement enforced by Hibernate. We can use any other extension of our convenience for this.

218. What are the steps for creating a SessionFactory in Hibernate?

Steps to create a SessionFactory in Hibernate are:

- Configuration: First create a Configuration object. This will refer to the path of configuration file.
- Resource: Add config file resource to Configuration object.
- Properties: Set properties in the Configuration object.
- SessionFactory: Use Configuration object to build SessionFactory.

Example:

Java

```
Configuration config = new Configuration();
config.addResource("testInstance/configuration.hbm.xml");
config.setProperties(System.getProperties());
SessionFactory sessions = config.buildSessionFactory();
```

219. Why do we use POJO in Hibernate?

POJO stands for Plain Old Java Objects. A POJO is a Java bean with getter and setter methods for each property of the bean. It is a simple class that encapsulates an object's properties and provides access through setters and getters.

Some of the reasons for using POJO in Hibernate are:

- POJO emphasizes the fact that this class is a simple Java class, not a heavy class like EJB.
- POJO is a well-constructed class, so it works well with Hibernate proxies.
- POJO also comes with a default constructor that makes it easier to persist with a default constructor.

220. What is Hibernate Query Language (HQL)?

Hibernate Query Language is also known as HQL. It is an Object Oriented language, but it is similar to SQL. HQL works well with persistent objects and their properties. HQL does not work on database tables. HQL queries are translated into native SQL queries specific to a database. HQL supports direct running of native SQL queries also, but it creates an issue in Database portability.

221. How will you call a stored procedure in Hibernate?

Hibernate supports executing not only simple queries but also stored procedures of the database. There are three ways to call a stored procedure in Hibernate:

XML mapping file:

We can declare the stored procedure inside the XML Mapping file.

```
1  <!-- Employee.hbm.xml -->
2  <hibernate-mapping>
3      <class name="com.testHibernate.util.Employee" table="employee">
4          <id name="employeeId" type="java.lang.Integer">
5              <column name="EMPLOYEE_ID" />
6              <generator class="identity" />
7          </id>
8
9          <property name="employeeId" type="string">
10             <column name="EMPLOYEE_ID" length="10" not-null="true" unique="true" />
11         </property>
12     </class>
13
14     <sql-query name="callEmployeeStoreProcedure">
15         <return alias="employee" class="com.testHibernate.util.Employee"/>
16         <![CDATA[CALL GetEmployees(:employeeId)]]>
17     </sql-query>
```

```

1 Query query = session.getNamedQuery("callEmployeeStoreProcedure")
2   .setParameter("employeeId", "1234");
3
4 List result = query.list();
5 for (int i = 0; i < result.size(); i++) {
6     Employee employee = (Employee) result.get(i);
7     System.out.println(employee.getEmployeeCode());
8 }

```

Native SQL:

We can use Native SQL to call a stored procedure query directly. In this example, GetEmployees() stored procedure is being called.

```

1 Query query = session.createSQLQuery(
2     "CALL GetEmployees(:employeeId)"
3     .addEntity(Employee.class)
4     .setParameter("employeeId", "1234");
5
6 List result = query.list();
7 for (int i = 0; i < result.size(); i++) {
8     Employee employee = (Employee) result.get(i);
9     System.out.println(employee.getEmployeeCode());

```

Use annotation:

We can also mark out stored procedure with @NamedNativeQueries annotation.

```

1 // Employee.java
2 import javax.persistence.Entity;
3 import javax.persistence.Table;
4 import javax.persistence.NamedNativeQueries;
5 import javax.persistence.NamedNativeQuery;
6
7 @NamedNativeQueries({
8     @NamedNativeQuery(
9         name = "callEmployeeStoreProcedure",
10        query = "CALL GetEmployees(:employeeId)",
11        resultClass = Employee.class
12    )
13 })
14 @Entity
15 @Table(name = "employee")
16 public class Employee implements java.io.Serializable {
17     // fields, getters, setters...
18
19     // Example field/method used in the print
20     private String employeeCode;
21
22     public String getEmployeeCode() {
23         return employeeCode;
24     }
25 }

```

Call with getNamedQuery()

```

1 // Java
2 Query query = session.getNamedQuery("callEmployeeStoreProcedure")
3   .setParameter("employeeId", "1234");
4
5 List result = query.list();
6 for (int i = 0; i < result.size(); i++) {
7     Employee employee = (Employee) result.get(i);
8     System.out.println(employee.getEmployeeCode());
9 }

```

222. What is Criteria API in Hibernate?

Criteria is a simplified API in Hibernate to get entities from the database by creating Criterion objects. It is a very intuitive and convenient approach for search features. Users can specify different criteria for searching entities and Criteria API can handle these. Criterion instances are obtained through factory methods on Restrictions.

223. Why do we use HibernateTemplate?

This is a trap question. HibernateTemplate has been deprecated. There were earlier good reasons to use HibernateTemplate. But now the trend has changed towards not using it anymore.

224. How can you see SQL code generated by Hibernate on console?

To display the SQL generated by Hibernate, we have to turn on the show_sql flag. This can be done in Hibernate configuration as follows:

XML

```
1 <property name="show_sql">true</property>
```

225. What are the different types of collections supported by Hibernate?

Hibernate supports the following two types of collections:

- Indexed Collections: List and Maps
- Sorted Collections: java.util.SortedMap and java.util.SortedSet

226. What is the difference between session.save() and session.saveOrUpdate() methods in Hibernate?

Save method first stores an object in the database. Then it persists the given transient instance by assigning a generated identifier. Finally, it returns the id of the entity that is just created.

SaveOrUpdate() method calls either save() or update() method. It selects one of these methods based on the existence of identifier. If an identifier exists for the entity then update() method is called. If there is no identifier for the entity then save() method is called as mentioned earlier.

227. What are the advantages of Hibernate framework over JDBC?

Main advantages of Hibernate over JDBC are as follows:

- Database Portability: Hibernate can be used with multiple types of database with easy portability. In JDBC, developer has to write database specific native queries. These native queries can reduce the database portability of the code.
- Connection Pool: Hibernate handles connection pooling very well. JDBC requires connection pooling to be defined by developer.
- Complexity: Hibernate handles complex query scenarios very well with its internal API like Criteria. So developer need not gain expertise in writing complex SQL queries. In JDBC application developer writes most of the queries.

228. How can we get statistics of a SessionFactory in Hibernate?

In Hibernate we can get the statistics of a SessionFactory by using Statistics interface. We can get information like Close Statement count, Collection Fetch count, Collection Load count, Entity insert count etc.

229. What is the Transient state of an object in Hibernate?

When an object is just instantiated using the new operator but is not associated with a Hibernate Session, then the object is in Transient state. In Transient state, object does not have a persistent representation in database. Also there is no identifier assigned to an object in Transient state. An object in Transient state can be garbage collected if there is no reference pointing to it.

230. What is the Detached state of an object in Hibernate?

An object is in detached state if it was persistent earlier but its Session is closed now. Any reference to this object is still valid. We can even update this object. Later on we can even attach an object in detached state to a new session and make it persistent. Detached state is very useful in application transactions where a user takes some time to finish the work.

231. What is the use of Dirty Checking in Hibernate?

Dirty Checking is a very useful feature of Hibernate for write to database operations. Hibernate monitors all the persistent objects for any changes. It can detect if an object has been modified or not. By Dirty Checking, only those fields of an object are updated that require any change in them. It reduces the time-consuming database write operations.

232. What is the purpose of Callback interface in Hibernate?

Callback interface in Hibernate is mainly used for receiving notifications of different events from an object. E.g. We can use Callback to get the notification when an object is loaded into or removed from database.

233. What are the different ORM levels in Hibernate?

There are following four different ORM levels in Hibernate:

- Pure Relational ORM: At this level entire application is designed around the relational model. All the operations are SQL based at this level.
- Light Object Mapping: At this level entity classes are mapped manually to relational tables. Business logic code is hidden from data access code. Applications with less number of entities use this level.
- Medium Object Mapping: In this case, application is designed around an object model. Most of the SQL code is generated at compile time. Associations between objects are supported by the persistence mechanism. Object-oriented expression language is used to specify queries.
- Full Object Mapping: This is one of the most sophisticated object modeling level. It supports composition, inheritance, polymorphism and persistence. The persistent classes do not inherit any special base class at this level. There are efficient fetching and caching strategies implemented transparently to the application.

234. What are the different ways to configure a Hibernate application?

There are mainly two ways to configure Hibernate application:

- XML based: We can define the Hibernate configuration in an XML file like hibernate.cfg.xml file.
- Programming based: We can also use code logic to configure Hibernate in our application.

235. What is Query Cache in Hibernate?

Hibernate provides Query Cache to improve the performance of queries that run multiple times with same parameters. At times Query Caching can reduce the performance of Transactional processing. By default Query Cache is disabled in Hibernate. It has to be used based on the benefits gained by it in performance of the queries in an application.

236. What are the different types of Association mappings supported by Hibernate?

Hibernate supports following four types of Association mappings:

- Unidirectional association: This kind of association works in only one direction.
- Unidirectional association with join tables
- Bidirectional association: This kind of association works in both directions.
- Bidirectional association with join tables

237. What are the different types of Unidirectional Association mappings in Hibernate?

In Hibernate there can be following three types of Unidirectional Association mappings:

- Many to one
- One to one
- One to many

238. What is Unit of Work design pattern?

Unit of Work is a design pattern to define business transactions. A Unit of Work is a list of ordered operations that we want to run on a database together. Either all of these go together or none of these goes. Most of the time, we use term business transaction in place of Unit of Work.

E.g. In case of money transfer from account A to B, the unit of work can be two operation Debit account A and Credit account B in a sequence. Both these operations should happen together and in right sequence.

239. In Hibernate, how can an object go in Detached state?

Once the session attached to an Object is closed, the object goes into Detached state. An Object in Detached state can be attached to another session at a later point of time. This state is quite useful in concurrent applications that have long unit of work.

240. How will you order the results returned by a Criteria in Hibernate?

Hibernate provides an Order criterion that can be used to order the results. This can be order objects based on their property in ascending or descending order. Class is org.hibernate.criterion.Order.

One example is as follows:

```
1 List employees = session.createCriteria(Employee.class)
2   .add(Restrictions.like("name", "F%"))           // Filter names starting with 'F'
3   .addOrder(Order.asc("name"))                     // Sort by name ascending
4   .addOrder(Order.desc("age"))                     // Then by age descending
5   .setMaxResults(10)                               // Limit to 10 results
6   .list();
```

241. How does Example criterion work in Hibernate?

In Hibernate, we can create an object with desired properties. Then we can use this object to search for objects with similar object. For this we can use org.hibernate.criterion.Example criterion.

E.g. First we create a sample book object of author Richard and category mystery. Then we search for similar books.

Java

```
1 Book book = new Book();
2 book.setAuthor("Richard");
3 book.setCategory(Category.MYSTERY);
4
5 List results = session.createCriteria(Book.class)
6   .add(Example.create(book))
7   .list();
```

242. How does Transaction management work in Hibernate?

In Hibernate we use Session interface to get a new transaction. Once we get the transaction we can run business operations in that transaction. At the end of successful business operations, we commit the transaction. In case of failure, we rollback the transaction.

Sample code is as follows:

```

1  Session s = null;
2  Transaction trans = null;
3
4  try {
5      s = sessionFactory.openSession();
6      trans = s.beginTransaction();
7
8      doTheAction(s); // Your business logic here
9
10     trans.commit();
11 } catch (RuntimeException exc) {
12     if (trans != null) {
13         trans.rollback();
14     }
15     throw exc; // Optionally rethrow or handle
16 } finally {
17     if (s != null) {
18         s.close();
19     }
20 }

```

243. How can we mark an entity/collection as immutable in Hibernate?

In Hibernate, by default an entity or collection is mutable. We can add, delete or update an entity/collection. To mark an entity/collection as immutable, we can use one of the following:

- @Immutable: We can use the annotation @Immutable to mark an entity/collection immutable.
- XML file: We can also set the property mutable=false in the XML file for an entity to make it immutable.

244. What are the different options to retrieve an object from database in Hibernate?

In Hibernate, we can use one of the following options to retrieve objects from database:

- Identifier: We can use load() or get() method and pass the identifier like primary key to fetch an object from database.
- HQL: We can create a HQL query and get the object after executing the query.
- Criteria API: We can use Criteria API to create the search conditions for getting the objects from database.
- Native SQL: We can write native SQL query for a database and just execute it to get the data we want and convert it into desired object.

245. How can we auto-generate primary key in Hibernate?

We can use the primary key generation strategy of type GenerationType.AUTO to auto-generate primary key while persisting an object in Hibernate.

Example:

Java

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private int id;

Show more lines

We can leave it null/0 while persisting and Hibernate automatically generates a primary key for us. Sometimes, AUTO strategy refers to a SEQUENCE instead of an IDENTITY.

246. How will you re-attach an object in Detached state in Hibernate?

We can call one of the methods Session.update(), Session.saveOrUpdate(), or Session.merge() to re-attach an object in detached state with another session in Hibernate.

247. What is the first level of cache in Hibernate?

A Hibernate Session is the first level of cache for persistent data in a transaction. The second level of cache is at JVM or SessionFactory level.

248. What are the different second level caches available in Hibernate?

In Hibernate, we can use different cache providers for implementing second level cache at JVM/SessionFactory level. Some of these are:

- Hashtable
- EHCache
- OSCache
- SwarmCache
- JBoss Cache 1.x
- JBoss Cache 2

249. Which is the default transaction factory in Hibernate?

In Hibernate, default transaction factory is JDBCTransactionFactory. But we can change it by setting the property `hibernate.transaction.factory_class`.

250. What are the options to disable second level cache in Hibernate?

This is a trick question. By default Second level cache is already disabled in Hibernate. In case, your project is using a second level cache you can use one of the following options to disable second level cache in Hibernate:

- We can set `hibernate.cache.use_second_level_cache` to false.
- We can use `CacheMode.IGNORE` to stop interaction between the session and second-level cache. Session will interact with cache only to invalidate cache items when updates occur.

251. What are the different fetching strategies in Hibernate?

Hibernate 3 onwards, there are the following fetching strategies to retrieve associated objects:

- Join fetching: Hibernate uses OUTER join to retrieve the associated instance or collection in the same SELECT.
- Select fetching: Hibernate uses a second SELECT to retrieve the associated entity or collection. You can explicitly disable lazy fetching by specifying `lazy="false"`. By default, lazy fetching is true.
- Subselect fetching: Hibernate uses a second SELECT to retrieve the associated collections for all entities retrieved in a previous query or fetch.
- Batch fetching: Hibernate uses a single SELECT to retrieve a batch of entity instances or collections by specifying a list of primary or foreign keys. This is a very good performance optimization strategy for select fetching.

252. What is the difference between Immediate fetching and Lazy collection fetching?

In Immediate fetching, an association, collection, or attribute is retrieved at the same time when the owner is loaded. But in Lazy collection fetching, a collection is fetched only when an operation is invoked on that collection by the client application. This is the default fetching strategy for collections in Hibernate. Lazy fetching is better from a performance perspective.

253. What is 'Extra lazy fetching' in Hibernate?

In Extra lazy fetching, only individual elements of a collection are fetched from the database when they are required. In this strategy, Hibernate does not fetch the whole collection into memory unless it is essential. It is a good fetching strategy for large collections of objects.

254. How can we check if a collection is initialized or not under Lazy Initialization strategy?

Hibernate provides two convenient methods, `Hibernate.initialize()` and `Hibernate.isInitialized()`, to check whether a collection is initialized or not. By using `Hibernate.initialize()`, we can force the initialization of a collection in Hibernate.

255. What are the different strategies for cache mapping in Hibernate?

Hibernate provides the following strategies for cache mapping:

- Read only: If an application requires caching only for read but not for write operations, then we can use this strategy. It is very simple to use and gives very good performance benefit. It is also safe to use in a cluster environment.
- Read/Write: If an application also needs caching for write operations, then we use Read/Write strategy. Read/write cache strategy should not be used if there is a requirement for serializable transaction isolation level. If we want to use it in a cluster environment, we need to implement a locking mechanism.
- Nonstrict Read/Write: If an application only occasionally updates the data, then we can use this strategy. It cannot be used in systems with serializable transaction isolation level requirement.
- Transactional: This strategy supports full transactional cache providers like JBoss TreeCache.

256. What is the difference between a Set and a Bag in Hibernate?

A Bag in Hibernate is an unordered collection. It can have duplicate elements. When we persist an object in a bag, there is no guarantee that the bag will maintain any order. A Set in Hibernate can only store unique objects. If we add the same element to a set a second time, it just replaces the old one. By default, a Set is an unordered collection in Hibernate.

257. How can we monitor the performance of Hibernate in an application?

We can use the following ways to monitor Hibernate performance:

- Monitoring SessionFactory: Since there is one SessionFactory in an application, we can collect the statistics of a SessionFactory to monitor performance. Hibernate provides sessionFactory.getStatistics() method to get the statistics of SessionFactory. Hibernate can also use JMX to publish metrics.
- Metrics: In Hibernate, we can also collect other metrics like number of open sessions, retrieved JDBC connections, cache hit, miss, etc. These metrics give great insight into the performance of Hibernate. We can tune Hibernate settings and strategies based on these metrics.

258. How can we check if an Object is in Persistent, Detached or Transient state in Hibernate?

We can use the following methods to check the state of an object in Hibernate:

- Persistent State: If call to EntityManager.contains(object) returns true, the object is in Persistent state.
- Detached State: If the call to PersistenceUnitUtil.getIdentifier(object) returns identifier property, then the object is in detached state.
- Transient State: If call to PersistenceUnitUtil.getIdentifier(object) returns null, then the object is in Transient state.

We can get access to PersistenceUnitUtil from the EntityManagerFactory in Hibernate.

259. What is 'the inverse side of association' in a mapping?

Let us consider an example in which a customer can have multiple orders and for every order there has to be a customer. In the OO world, the customer is the owner of the order. In the SQL world, an Order has a reference to customer id. It is a bi-directional one-to-many mapping from customer to order. The inverse side in this mapping is the owner of the object. In this case, the customer is the owner of the order, since an order cannot exist without a customer. But a customer can exist without an order. Also, the customer has no column to save order data. But an Order table can store customer id, which is used for mapping.

260. What is ORM metadata?

ORM uses metadata for its internal work. ORM maintains metadata to generate code used for accessing columns and tables. ORM maps classes to tables and stores this information in Metadata. It maps fields in classes to columns in tables. These kinds of mappings are also part of Metadata. Application developers can also access Hibernate Metadata by using ClassMetadata and CollectionMetadata interfaces and Type hierarchy.

261. What is the difference between load() and get() method in Hibernate?

In Hibernate, load() and get() methods are quite similar in functionality. The main difference is that load() method will throw an ObjectNotFoundException if the row corresponding to an object is not found in the database. On the other

hand, `get()` method returns null value when an object is not found in the database. It is recommended that we should use `load()` method only when we are sure that the object exists in the database.

262. When should we use `get()` method or `load()` method in Hibernate?

As a thumb rule, we can follow these guidelines:

- We should use `get()` method when we want to load an object.
- We should use `load()` method when we need a reference to an object without running extra SQL queries.

263. What is a derived property in Hibernate?

In Hibernate, a derived property is not mapped to any column of a database table. A derived property is computed at runtime by evaluation of an expression. These are read-only properties. For example, in this example `profitMargin` is derived from `salePrice` and `buyPrice`:

```
1 <property name="profitMargin"
2     formula="(SELECT (i.salePrice - i.buyPrice) FROM item i WHERE i.Id = Id)" />
```

264. How can we use Named Query in Hibernate?

A Named SQL query is the HQL query that is associated with a string name and can be referenced in the application by name. It can be used in the following ways:

XML Mapping File: We can define it in XML mapping file.

```
1 <query name="findBookByAuthor">
2     <![CDATA[from Book s where s.author = :author]]>
3 </query>
```

Annotation: We can also mark Named SQL with annotation.

```
1 @NamedQueries({
2     @NamedQuery(
3         name = "findBookByAuthor",
4         query = "from Book s where s.author = :author"
5     )
6 })
7 @Entity
8 @Table(name = "book")
9 public class Book implements java.io.Serializable {
10     // fields, getters, setters...
11 }
```

265. What are the two locking strategies in Hibernate?

There are two popular locking strategies that can be used in Hibernate:

- **Optimistic:** In Optimistic locking, we assume that multiple transactions can complete without affecting each other. So we let the transactions do their work without locking the resources initially. Just before the commit, we check if any of the resources have changed by another transaction, then we throw an exception and rollback the transaction.
- **Pessimistic:** In Pessimistic locking, we assume that concurrent transactions will conflict while working with the same resources. So a transaction has to first obtain a lock on the resources it wants to update. The other transaction can proceed with the same resource only after the lock has been released by the previous transaction.

266. What is the use of version number in Hibernate?

Version number is used in optimistic locking in Hibernate. When a transaction modifies an object, it increments its version. Based on the version number, a second transaction can determine if the object it has read earlier has changed or not. If the version number at the time of write is different than the version number at the time of read, then we should not commit the transaction.

267. What is the use of session.lock() method in Hibernate?

Session.lock() is a deprecated method in Hibernate. We should not use it. Instead, we should call buildLockRequest(LockMode).lock(entityName, object) method in Hibernate.

268. What inheritance mapping strategies are supported by Hibernate?

Hibernate supports the following inheritance mapping strategies between classes and tables:

- Table per class hierarchy: In case of multiple types of books, we can have one book class and one book table. We can store all child classes of book like HardcoverBook, PaperbackBook, etc., in the same table book. But we can identify the subclasses by a BookType column in the Book table.
- Table per subclass: In this case, we can have a separate table for each kind of book. HardcoverBook table for HardcoverBook book class. PaperbackBook table for PaperbackBook book class. And there will be a parent table, Book for Book class.
- Table per concrete class: In this case, we also have a separate table for each kind of book. But in this case, we have even inherited properties defined inside each table. There is no parent table Book for Book class, since it is not a concrete class.

269. What is Maven?

Maven is a software project management tool. It is open source software from Apache Software Foundation. It is used for building, reporting and documenting a software project. It is mainly based on POM (Project Object Model).

270. What are the main features of Maven?

Some of the main features of Maven are:

- Simple: Maven provides simple project setup that is based on best practices.
- Fast: You can get a new project or module started in a few seconds in Maven.
- Easy to learn: Maven usage and commands are easy to learn across all projects. Therefore ramp up time for new developers coming onto a project is very less.
- Dependency management: Maven provides superior dependency management including automatic updates and transitive dependencies.
- Multiple Projects: You can easily work with multiple projects at the same time by using Maven.
- Large Library: Maven has a large and growing repository of libraries and metadata to use out of the box.
- Extensible: Maven supports the ability to easily write plugins in Java or scripting languages for extending its core functionality.
- Instant: Maven is online and it provides instant access to new features with very less configuration.

271. What areas of a Project can you manage by using Maven?

Maven can help us manage the following areas of a project:

- Build
- Testing
- Release
- Reporting
- Software Change Management (SCM)
- Documentation

- Distribution

272. What are the main advantages of Maven?

Maven has a long list of advantages for software development. Some of the main advantages are:

- **Common Project Structure:** By using Maven, every developer has a common project structure that helps in understanding the code as well as developing new features in a new project.
- **Modular Design:** Maven promotes modular design that divides a complex project into multiple modules that are easier to manage. By using Maven, it is easier to manage multiple modules for build, test, release etc.
- **Centralized Dependency Management:** With Maven, each developer does not have to include the jars separately in each project or module. Maven provides a centralized dependency management that can help improve efficiency of software development.
- **Fewer Decisions:** With Maven a developer has to make fewer decisions about things unrelated to software development work. The project structure comes ready with Maven, dependency management is a uniform approach and build/release are handled by Maven. So a developer can focus on core work of developing software.

273. Why do we say “Maven uses convention over configuration”?

Convention over configuration is a Software Design Paradigm that decreases the number of decisions made by a software developer, without losing flexibility. In Maven, there are many conventions for setting up the project, building the artifacts, running unit tests and releasing the code. These conventions lead to common process for Software development. In case of other tools, there are a lot of configuration options present. But most of the time, a developer uses same set of configuration options. So it is better to make these as default options. Maven uses default options from best practices and provides right conventions for Software development.

274. What are the responsibilities of a Build tool like Maven?

A Build tool like Maven helps us with the following tasks:

- **Source Code:** A Build tool can generate source code based on templates.
- **Documentation:** We can get documentation files from source code by using a build tool. E.g. Javadoc
- **Compilation:** Primary responsibility of a Build tool is to compile source code into executable code.
- **Packaging:** A Build tool packages compiled code into a deployable file like jar, zip, war etc.
- **Deployment:** We can deploy the packaged code on server by using a Build tool.

275. What are the differences between Ant and Maven?

Key differences between Ant and Maven are:

- Ant is a Java library and command line toolbox for build process. Maven is a framework for many aspects of software development like project setup, compile, build, documentation etc.
- Ant does not have any conventions for project structure or build processes. Maven has conventions for setting up project structure as well as for build processes.
- Ant is based on procedural programming. We have to write code for compilation, build, copy etc. Maven is based on declarative programming. We have to just configure it for our project setup and programming.
- Ant does not impose any lifecycle. We need to create the sequence of tasks manually. Maven has a lifecycle for software build processes. There are well-defined phases that we can use in Maven.
- Ant scripts are not reusable in multiple projects. Maven has plugins that are reusable across multiple projects.

276. What is MOJO in Maven?

MOJO stands for Maven plain Old Java Object. Every MOJO is an executable goal in Maven. It is like an annotated Java class. It specifies metadata about a goal like goal name, phase of lifecycle for goal and parameters required by goal. A Maven plugin can contain multiple MOJOs.

277. What is a Repository in Maven?

A repository is a location on file system where build artifacts, jars, dependencies and pom.xml files are stored.

278. What are the different types of repositories in Maven?

There are mainly two types of repositories in Maven:

- Local Repository: This is your local folder in which a copy of your installation and dependencies is stored.
- Remote Repository: This is a remote folder in which jars and other build artifacts are stored. These can be located on servers within your organization.
- Central Remote Repository: This is the central Maven repository that is located on repo.maven.apache.org or uk.maven.org or any other third party location. This is where we can find artifacts from different providers that are available for download and use. Like Hibernate, Spring libraries etc.

279. What is a local repository in Maven?

Maven local repository is a folder in your local file system in which your project's installation, dependency jars, plugins etc. are stored. Default location of Maven local repository is .m2 folder. It can be located under following location on file system:

- Windows

```
C:\Documents and Settings\\.m2
```

- Unix/Linux/Mac

```
~/ .m2
```

280. What is a central repository in Maven?

Maven central repository is a truly remote repository that is located on repo.maven.apache.org or uk.maven.org or any other third party location. This contains the jars and artifacts provided by various software providers. Central repository contains a large amount of data. Therefore it is not allowed to scrape the whole site. But you can use the relevant jars that you want for download and use in your Maven project.

281. What is a Remote repository in Maven?

A Remote repository is a remote location on the internet where the jars and dependencies from different vendors are stored. These files can be accessed by protocols like `file://` or `http://` etc. These can be truly remote repositories set up by third party vendors or locations inside your organization that contains the relevant jars required by your project.

282. Why should we not store jars in CVS or any other version control system instead of Maven repository?

Maven recommends storing jars in local repository instead of CVS or any other version control system. There are following advantages of storing it in Maven repo vs. CVS:

- Less Storage: A repository is very large, but it takes less space because each JAR is stored only in one place. E.g. If we have 10 modules dependent on Spring jar, then they all refer to same Spring jar stored in local repository.
- Quicker Checkout: Project checkout is quicker from local repository, since there is not need to checkout jars if they are already present in repo.
- No need for versioning: There is no need to version JARS since external dependencies do not change so often.

283. Can anyone upload JARS or artifacts to Central Repository?

No, we need special permissions to upload JARS and artifacts to Central Maven Repository.

284. What is a POM?

POM is an abbreviation for Project Object Model. This is the basic unit of work in Maven. It is an XML file with name `pom.xml`. It contains details of project and project configuration that are used by Maven to build the project. It also contains default values for many projects. E.g. `target` is the name of build directory for Java Maven project.

285. What is Super POM?

Super POM is Maven's default POM. All the POM files extend from Super POM.

286. What are the main required elements in POM file?

Every POM file should have the following required elements:

- project root
- modelVersion
- groupId: the id of the project's group.
- artifactId: the id of the artifact (project)
- version: the version of the artifact under the specified group

287. What are the phases in Build lifecycle in Maven?

In Maven, each build lifecycle consists of many phases. Default build lifecycle has the following phases:

- validate: In this phase, Maven validates that the project is correct and all necessary information is available to run next phase.
- compile: Maven compiles the source code of the project in this phase.
- test: This is the phase to run unit tests on the compiled source. There should not be any need to package or deploy the code to run these tests.
- package: In this phase, Maven takes the compiled code and packages it in its distributable format, such as a JAR.
- verify: Maven runs any checks on results of integration tests to ensure that quality criteria are met.
- install: In this phase, Maven installs the package into local repository. After this it can be used as a dependency in other projects locally.
- deploy: In the build environment, Maven copies the final package to the remote repository for sharing with other developers and projects.

288. What command will you use to package your Maven project?

To package a project into a distributable format we use the following command:

```
mvn -package
```

289. What is the format of fully qualified artifact name of a Maven project?

A Maven project has artifact name with the following format:

::

Following is the convention used by some organizations:

Parent pom

groupId: org.Orgname.Projectname

artifactId: org.Orgname.Projectname

version: x.x.x

E.g. org.Orgname.Projectname:org.Orgname.Projectname-1.0.0.pom

Modules

groupId: org.Orgname.Projectname

artifactId: org.Orgname.Projectname.Modulename

version: x.x.x

E.g. org.Orgname.Projectname:org.Orgname.Projectname.Modulename-1.0.0.jar

290. What is an Archetype in Maven?

As per official definition, an Archetype is a Maven project templating toolkit. By using an Archetype, an author of Archetype can create a Project template. Users of this project template (archetype) can pass different parameters to this template and start using it. Archetype promotes consistency in the process of creating and working on a project. It also helps in reducing the ramp up time for new developers to come on board on a project.

291. What is the command in Maven to generate an Archetype?

In Maven, we can use the following command to generate an Archetype:

```
mvn archetype:generate
```

292. What are the three main build lifecycles of Maven?

Maven has the following three build lifecycles that further contain multiple phases:

- **clean:** In this lifecycle any files generated by previous builds are removed.
- **default:** This lifecycle is used for validating, compiling and creating the application. It has multiple phases like compile, test, package inside it.
- **site:** Maven generates and deploys the documentation of a site in this phase.

293. What are the main uses of a Maven plugin?

Maven is mainly a plugin execution framework. At the core of Maven all the work is done by plugins. A Maven plugin can be used for the following purposes:

- Cleaning up the code
- Compiling the code
- Creating a JAR file
- Deploying the artifacts
- Running the unit tests
- Documenting the project
- Generating the site of a project
- Generating a WAR file
- Generate a checkstyle report

294. How will you find the version of a plugin being used?

Maven Help Plugin has a describe goal. This can be used for listing the version of a plugin. Sample command for this is:

```
mvn -Dplugin=install help:describe
```

Note: In the above command replace Dplugin with the plugin prefix as the argument. Do not use the artifact ID of plugin here.

295. What are the different types of profile in Maven? Where will you define these profiles?

In Maven, we can have the following types of Profile:

- **Per Project:** It is defined in the POM itself (pom.xml).
- **Per User:** We can define it in the Maven-settings (%USER_HOME%/.m2/settings.xml).
- **Global:** It is defined in the global Maven-settings (\${maven.home}/conf/settings.xml).
- **Profile descriptor:** Descriptor is located in project basedir (profiles.xml) (It is not supported in Maven 3.0)

296. What are the different setting files in Maven? Where will you find these files?

Maven is very simple to use. At the core it has a setting file named settings.xml. This file contains the setting element that is used to configure the Maven with different options. The main locations where this file can be found are:

- **Maven Installation directory:** \${maven.home}/conf/settings.xml
- **User Home directory:** \${user.home}/.m2/settings.xml

297. What are the main elements we can find in settings.xml?

In settings.xml we can have all the configuration information for Maven. Some of the important elements are:

- **localRepository:** The value of this element is the path of this build system's local repository. The default value is \${user.home}/.m2/repository. It is used for a main build server to allow all logged-in users to build from a common local repository.

- **interactiveMode:** If it is true then Maven should attempt to interact with the user for input. If it is false then Maven does not interact with the user. Default setting is true.
- **usePluginRegistry:** If it is true Maven uses the `${user.home}/.m2/plugin-registry.xml` file to manage plugin versions. By default it is false.
- **offline:** If it is true this build system should be able to operate in offline mode. By default it is false. This element is used for build servers that cannot connect to a remote repository due to network setup or security reasons.

298. How will you check the version of Maven in your system?

We can use the following command in console to check the version of Maven in our system:

```
mvn -version
```

299. How will you verify if Maven is installed on Windows?

To check this, type `mvn -version` in cmd prompt of Windows. This will give you the version of Maven installed on Windows.

300. What is a Maven artifact?

A Maven artifact is a file that gets deployed to a Maven repository. In most cases it is a JAR file. When Maven build runs, it creates one or more artifacts. In case of Java projects, it produces a compiled jar and a sources jar. Every artifact in Maven has a group ID, an artifact ID and a version string. These three attributes uniquely identify an artifact. In Maven, we specify a project's dependencies as artifacts.

301. What are the different dependency scopes in Maven?

Maven supports the following dependency scopes:

- **compile:** This is the default dependency scope in Maven. The compile level dependencies are available in all classpaths of a project. These dependencies are also propagated to dependent projects.
- **provided:** This scope is similar to compile. But in this scope we expect the JDK or a container to provide the dependency at runtime. E.g. While building a web application for the Java Enterprise Edition, we can set the dependency on the Servlet API and related Java EE APIs to scope provided. The web container will provide these classes at runtime to our application. This scope is only available on the compilation and test classpath, and is not transitive.
- **runtime:** The dependency in this scope is not required for compilation. It is required for execution. It is available in the runtime and test classpaths. It is not present in the compile classpath.
- **test:** This scope is used for dependencies that are required for test compilation and execution phases. This scope is not transitive.
- **system:** This scope is same as provided scope, except that you have to provide the JAR that contains it explicitly. In this case, the artifact is always available. There is no need to look it up in a repository.
- **import:** This scope is only used on a dependency of type pom in the section. In this case, the specified POM has to be replaced with the dependencies in that POM's section. This scope is only available in Maven 2.0.9 or later.

302. How can we exclude a dependency in Maven?

To exclude a dependency we can add the tag under the section of the pom.

XML

```
<dependencies>
  <dependency>
    <groupId>test.ProjectX</groupId>
    <artifactId>ProjectX</artifactId>
    <version>1.0</version>
```

```
<scope>compile</scope>
<exclusions>
    <exclusion>
        <!-- exclusion is mentioned here -->
        <groupId>test.ProjectY</groupId>
        <artifactId>ProjectY</artifactId>
    </exclusion>
</exclusions>
</dependency>
</dependencies>
```

303. How Maven searches for JAR corresponding to a dependency?

Maven first looks for a JAR related to a dependency in the local repository. If it finds it there then it stops. If it does not find it in local repo, it looks for the JAR in the remote repository and downloads the corresponding version of JAR file. From remote repository it stores the JAR into local repository.

304. What is a transitive dependency in Maven?

Let's say you have a Project A that depends on dependency B. The dependency B further depends on dependency C. So your dependency C is a Transitive Dependency of your project A. In Maven, starting from 2.0, you do not have to specify transitive dependencies. You just mention your immediate dependencies in pom.xml. Maven takes care of resolving the Transitive dependencies and includes them automatically.

305. What are Excluded dependencies in Maven?

Let's say a project A depends on project B, and project B depends on project C. The developers of project A can explicitly exclude project C as a dependency. We can use the "exclusion" element to exclude it. Such dependencies are called Excluded dependencies in Maven.

306. What are Optional dependencies in Maven?

Let's say a project B depends on project C. The developers of project B can mark project C as an optional dependency by using the "optional" element. In case project A depends on project B, A will depend only on B and not on B's optional dependency C. The developers of project A may then explicitly add a dependency on C. The dependency of B on C is known as Optional dependency in Maven.

307. Where will you find the class files after compiling a Maven project successfully?

Once Maven completes the compilation successfully, it stores the files in target folder. The default location for class files is:

```
${basedir}/target/classes/
```

308. What are the default locations for source, test and build directories in Maven?

The default locations are as follows:

Source: src/main/java

Test: src/main/test

Build: Target

309. What is the result of jar:jar goal in Maven?

In Maven, jar:jar goal creates a jar file in the Maven build directory. Jar file is created with the name format \${project.id}-\${project.currentVersion}.jar. The id and currentVersion are mentioned in the project.xml of the project being built. jar:jar does not recompile sources. It just creates a jar from already compiled classes.

310. How can we get the debug or error messages from the execution of Maven?

At times, project build or compile fails in Maven. At this time it is very helpful to see the debug or error messages from Maven execution. To get the debug messages we can call Maven with -X option. To get the error/exception messages we can call Maven with -e option.

311. What is the difference between a Release version and SNAPSHOT version in Maven?

A SNAPSHOT version in Maven is the one that has not been released. Before every release version there is a SNAPSHOT version. Before 1.0 release there will be 1.0-SNAPSHOT. If we download 1.0-SNAPSHOT today then we may get different set of files than the one we get on downloading it yesterday. SNAPSHOT version can keep getting changes in it since it is under development. But release version always gives exactly same set files with each download.

312. How will you run test classes in Maven?

We need Surefire plugin to run the test classes in Maven. To run a single test we can call following command:

Shell

```
mvn -Dtest=TestCaseA test
```

Show more lines

We can also use patterns to run multiple test cases:

Shell

```
mvn -Dtest=TestCase* test
```

Show more lines

or

Shell

```
mvn -Dtest=TestCaseA,TestCaseB,TestImportant* test
```

Show more lines

313. Sometimes Maven compiles the test classes but doesn't run them? What could be the reason for it?

In Maven, Surefire plugin is used for running the Tests. We can configure it to run certain test classes. Sometimes you may have unintentionally specified an incorrect value to \${test} in settings.xml or pom.xml. We need to look for following in pom.xml/settings.xml and fix it:

```
1 <properties>
2   <property>
3     <name>test</name>
4     <value>some-value</value>
5   </property>
6 </properties>
```

314. How can we skip the running of tests in Maven?

We can use the parameter `-Dmaven.test.skip=true` or `-DskipTests=true` in the command line for skipping the tests. The parameter `-Dmaven.test.skip=true` skips the compilation of tests. The parameter `-DskipTests=true` skips the execution of tests. Surefire plugin of Maven honors these parameters.

315. Can we create our own directory structure for a project in Maven?

Yes, Maven gives us the flexibility of creating our own directory structure. We just need to configure the elements like `src`, `test`, etc. in the section of `pom.xml`.

316. What are the differences between Gradle and Maven?

Gradle is nowadays getting more popular. Google uses it for Android development and release. Companies like LinkedIn also use Gradle. Gradle is based on Domain Specific Language (DSL). Maven is based on XML. Gradle gives more flexibility to do custom tasks similar to ANT. Maven scripts have predefined structure. So it is less flexible. Maven is mainly used for Java based systems. Gradle is used for a variety of languages. It is a Polyglot build tool.

317. What is the difference between Inheritance and Multi-module in Maven?

In Maven, we can create a parent project that will pass its values to its children projects. A multi-module project is created to manage a group of other sub-projects or modules. The multi-module relationship is like a tree that starts from the topmost level to the bottom level. In a multi-module project, we specify that a project should include the specific modules for build. Multi-module builds are used to group modules together in a single build. Whereas in Inheritance, the parent-child project relationship starts from the leaf node and goes upwards. It deals more with the definition of a specific project. In this case a child's pom is derived from its parent's pom.

318. What is Build portability in Maven?

In Maven, the portability of a build is the measure of how easy it is to take a particular project and build it in different environments. A build that does not require any custom configuration or customization of properties files is more portable than a build that requires a lot of custom work to build it from scratch. Open source projects from Apache Commons are one of the most portable projects. These builds can work just out of the box.

319. How can we see n most recent commits in GIT?

We can use `git log` command to see the latest commits. To see the three most recent commits we use the following command:

Shell

```
git log -3
```

320. How can we know if a branch is already merged into master in GIT?

We can use the following commands for this purpose:

- `git branch --merged master` : This prints the branches merged into master
- `git branch --merged` : This prints the branches merged into HEAD (i.e. tip of current branch)
- `git branch --no-merged` : This prints the branches that have not been merged

By default this applies only to local branches. We can use `-a` flag to show both local and remote branches. Or we can use `-r` flag to show only the remote branches.

321. What is the purpose of git stash drop?

In case we do not need a specific stash, we use git stash drop command to remove it from the list of stashes. By default, this command removes the latest added stash. To remove a specific stash we specify as argument in the git stash drop command.

322. What is the HEAD in GIT?

A HEAD is a reference to the currently checked out commit. It is a symbolic reference to the branch that we have checked out. At any given time, one head is selected as the 'current head'. This head is also known as HEAD (always in uppercase).

323. What is the most popular branching strategy in GIT?

There are many ways to do branching in GIT. One of the popular ways is to maintain two branches:

- master: This branch is used for production. In this branch HEAD is always in production ready state.
- develop: This branch is used for development. In this branch we store the latest code developed in project. This is work in progress code.

Once the code is ready for deployment to production, it is merged into master branch from develop branch.

324. What is SubGit?

SubGit is a software tool used for migrating SVN to Git. It is very easy to use. By using this we can create a writable Git mirror of a Subversion repository. It creates a bi-directional mirror that can be used for pushing to Git as well as committing to Subversion. SubGit also takes care of synchronization between Git and Subversion.

325. What is the use of git instaweb?

Git-instaweb is a script by which we can browse a git repository in a web browser. It sets up the gitweb and a web-server that makes the working repository available online.

326. What are git hooks?

Git hooks are scripts that can run automatically on the occurrence of an event in a Git repository. These are used for automation of workflow in GIT. Git hooks also help in customizing the internal behavior of GIT. These are generally used for enforcing a GIT commit policy.

327. What is GIT?

GIT is a mature Distributed Version Control System (DVCS). It is used for Source Code Management (SCM). It is open source software. It was developed by Linus Torvalds, the creator of Linux operating system. GIT works well with a large number of IDEs (Integrated Development Environments) like Eclipse, IntelliJ etc. GIT can be used to handle small and large projects.

328. What is a repository in GIT?

A repository in GIT is the place in which we store our software work. It contains a sub-directory called .git. There is only one .git directory in the root of the project. In .git, GIT stores all the metadata for the repository. The contents of .git directory are of internal use to GIT.

329. What are the main benefits of GIT?

There are following main benefits of GIT:

- **Distributed System:** GIT is a Distributed Version Control System (DVCS). So you can keep your private work in version control but completely hidden from others. You can work offline as well.
 - **Flexible Workflow:** GIT allows you to create your own workflow. You can use the process that is suitable for your project. You can go for centralized or master-slave or any other workflow.
 - **Fast:** GIT is very fast when compared to other version control systems.
 - **Data Integrity:** Since GIT uses SHA1, data is not easier to corrupt.
 - **Free:** It is free for personal use. So many amateurs use it for their initial projects. It also works very well with large size project.
 - **Collaboration:** GIT is very easy to use for projects in which collaboration is required. Many popular open source software across the globe use GIT.
-

330. What are the disadvantages of GIT?

GIT has very few disadvantages. These are the scenarios when GIT is difficult to use. Some of these are:

- **Binary Files:** If we have a lot binary files (non-text) in our project, then GIT becomes very slow. E.g. Projects with a lot of images or Word documents.
 - **Steep Learning Curve:** It takes some time for a newcomer to learn GIT. Some of the GIT commands are non-intuitive to a fresher.
 - **Slow remote speed:** Sometimes the use of remote repositories is slow due to network latency. Still GIT is better than other VCS in speed.
-

331. What are the main differences between GIT and SVN?

The main differences between GIT and SVN are:

- **Decentralized:** GIT is decentralized. You have a local copy that is a repository in which you can commit. In SVN you have to always connect to a central repository for check-in.
 - **Complex to learn:** GIT is a bit difficult to learn for some developers. It has more concepts and commands to learn. SVN is much easier to learn.
 - **Unable to handle Binary files:** GIT becomes slow when it deals with large binary files that change frequently. SVN can handle large binary files easily.
 - **Internal directory:** GIT creates only .git directory. SVN creates .svn directory in each folder.
 - **User Interface:** GIT does not have good UI. But SVN has good user interfaces.
-

332. How will you start GIT for your project?

We use git init command in an existing project directory to start version control for our project. After this we can use git add and git commit commands to add files to our GIT repository.

333. What is git clone in GIT?

In GIT, we use git clone command to create a copy of an existing GIT repository in our local. This is the most popular way to create a copy of the repository among developers. It is similar to svn checkout. But in this case the working copy is a full-fledged repository.

334. How will you create a repository in GIT?

To create a new repository in GIT, first we create a directory for the project. Then we run 'git init' command. Now, GIT creates .git directory in our project directory. This is how our new GIT repository is created.

335. What are the different ways to start work in GIT?

We can start work in GIT in following ways:

- New Project: To create a new repository we use git init command.
- Existing Project: To work on an existing repository we use git clone command.

336. GIT is written in which language?

Most of the GIT distributions are written in C language with Bourne shell. Some of the commands are written in Perl language.

337. What does 'git pull' command in GIT do internally?

In GIT, git pull internally does a git fetch first and then does a git merge. So pull is a combination of two commands: fetch and merge. We use git pull command to bring our local branch up to date with its remote version.

338. What does 'git push' command in GIT do internally?

In GIT, git push command does following two commands:

1. fetch: First GIT, copies all the extra commits from server into local repo and moves origin/master branch pointer to the end of commit chain.
2. merge: Then it merges the origin/master branch into the master branch. Now the master branch pointer moves to the newly created commit. But the origin/master pointer remains there.

339. What is git stash?

In GIT, sometimes we do not want to commit our code but we do not want to lose also the unfinished code. In this case we use git stash command to record the current state of the working directory and index in a stash. This stores the unfinished work in a stash, and cleans the current branch from uncommitted changes. Now we can work on a clean working directory. Later we can use the stash and apply those changes back to our working directory. At times we are in the middle of some work and do not want to lose the unfinished work, we use git stash command.

340. What is the meaning of 'stage' in GIT?

In GIT, stage is a step before commit. To stage means that the files are ready for commit. Let's say, you are working on two features in GIT. One of the features is finished and the other is not yet ready. You want to commit and leave for home in the evening. But you can commit since both of them are not fully ready. In this case you can just stage the feature that is ready and commit that part. Second feature will remain as work in progress.

341. What is the purpose of git config command?

We can set the configuration options for GIT installation by using git config command.

342. How can we see the configuration settings of GIT installation?

We can use 'git config --list' command to print all the GIT configuration settings in GIT installation.

343. How will you write a message with commit command in GIT?

We call following command for commit with a message:

Shell

```
$/> git commit -m <message>
```

344. What is stored inside a commit object in GIT?

GIT commit object contains following information:

- SHA1 name: A 40 character string to identify a commit
 - Files: List of files that represent the state of a project at a specific point of time
 - Reference: Any reference to parent commit objects
-

345. How many heads can you create in a GIT repository?

There can be any number of heads in a repository. By default there is one head known as HEAD in each repository in GIT.

346. Why do we create branches in GIT?

If we are simultaneously working on multiple tasks, projects, defects or features, we need multiple branches. In GIT we can create a separate branch for each separate purpose. Let's say we are working on a feature, we create a feature branch for that. In between we get a defect to work on then we create another branch for defect and work on it. Once the defect work is done, we merge that branch and come back to work on feature branch again. So working on multiple tasks is the main reason for using multiple branches.

347. What are the different kinds of branches that can be created in GIT?

We can create different kinds of branches for following purposes in GIT:

- Feature branches: These are used for developing a feature.
 - Release branches: These are used for releasing code to production.
 - Hotfix branches: These are used for releasing a hotfix to production for a defect or emergency fix.
-

348. How will you create a new branch in GIT?

We use following command to create a new branch in GIT:

Shell

```
$/> git checkout -b <branchname>
```

349. How will you add a new feature to the main branch?

We do the development work on a feature branch that is created from master branch. Once the development work is ready we use git merge command to merge it into master branch.

350. What is a pull request in GIT?

A pull request in GIT is the list of changes that have been pushed to GIT repository. Generally these changes are pushed in a feature branch or hotfix branch. After pushing these changes we create a pull request that contains the changes between master and our feature branch. This pull request is sent to reviewers for reviewing the code and then merging it into develop or release branch.

351. What is merge conflict in GIT?

A merge conflict in GIT is the result of merging two commits. Sometimes the commit to be merged and current commit have changes in the same location. In this scenario, GIT is not able to decide which change is more important. Due to

this, GIT reports a merge conflict. It means merge is not successful. We may have to manually check and resolve the merge conflict.

352. How can we resolve a merge conflict in GIT?

When GIT reports a merge conflict in a file, it marks the lines as follows:

E.g.

the business days in this week are

```
<<<<<<< HEAD
```

```
five
```

```
=====
```

```
six
```

```
>>>>>>> branch-feature
```

To resolve the merge conflict in a file, we edit the file and fix the conflicting change. In the above example, we can either keep five or six. After editing the file, we run git add command followed by git commit command. Since GIT is aware that it was a merge conflict, it links this change to the correct commit.

353. What command will you use to delete a branch?

After the successful merge of a feature branch in the main branch, we do not need the feature branch. To delete an unwanted branch, we use the following command:

```
git branch -d <branchname>
```

354. What command will you use to delete a branch that has unmerged changes?

To forcibly delete an unwanted branch with unmerged changes, we use the following command:

```
git branch -D <branchname>
```

355. What is the alternative command to merging in GIT?

Another alternative to merging in GIT is rebasing. It is done by the git rebase command.

356. What is Rebasing in GIT?

Rebasing is the process of moving a branch to a new base commit. It is like rewriting the history of a branch. In Rebasing, we move a branch from one commit to another. By this, we can maintain linear project history. Once the commits are pushed to a public repository, it is not a good practice to use Rebasing.

357. What is the 'Golden Rule of Rebasing' in GIT?

The golden rule of Rebasing is that we should never use git rebase on public branches. If other people are using the same branch, then they may get confused by looking at the changes in the Master branch after GIT rebasing. Therefore, it is not recommended to do rebasing on a public branch that is also used by other collaborators.

358. Why do we use Interactive Rebasing in place of Auto Rebasing?

By using Interactive rebasing, we can alter the commits before moving them to a new branch. This is more powerful than an automated rebase. It gives us complete control over the branch's commit history. Generally, we use Interactive Rebasing to clean up the messy history of commits just before merging a feature branch into master.

359. What is the command for Rebasing in Git?

Git command for rebasing is:

```
git rebase <new-commit>
```

360. What is the main difference between git clone and git remote?

The main difference between git clone and git remote is that git clone is used to create a new local repository whereas git remote is used in an existing repository. git remote adds a new reference to an existing remote repository for tracking further changes. git clone creates a new local repository by copying another repository from a URL.

361. What is GIT version control?

GIT version control helps us in managing the changes to source code over time by a software team. It keeps track of all the changes in a special kind of database. If we make a mistake, we can go back in time and see previous changes to fix the mistake. GIT version control helps the team in collaborating on developing software and working efficiently. Everyone can merge the changes with confidence that everything is tracked and remains intact in GIT version control. Any bug introduced by a change can be discovered and reverted back by going back to a working version.

362. What GUI do you use for working on GIT?

There are many GUIs for GIT that we can use. Some of these are:

- GitHub Desktop
 - GITX-dev
 - Gitbox
 - Git-cola
 - SourceTree
 - Git Extensions
 - SmartGit
 - GitUp
-

363. What is the use of git diff command in GIT?

In GIT, git diff command is used to display the differences between two versions, or between the working directory and an index, or between the index and the most recent commit. It can also display changes between two blob objects, or between two files on disk in GIT. It helps in finding the changes that can be used for code review for a feature or bug fix.

364. What is git rerere?

In GIT, rerere is a hidden feature. The full form of rerere is “reuse recorded resolution”. By using rerere, GIT remembers how we’ve resolved a hunk conflict. The next time GIT sees the same conflict, it can automatically resolve it for us.

365. What are the three most popular versions of git diff command?

Three most popular git diff commands are as follows:

- git diff: It displays the differences between the working directory and the index.
 - git diff –cached: It displays the differences between the index and the most recent commit.
 - git diff HEAD: It displays the differences between the working directory and the most recent commit.
-

366. What is the use of git status command?

In GIT, git status command mainly shows the status of the working tree. It shows the following items:

1. The paths that have differences between the index file and the current HEAD commit.
2. The paths that have differences between the working tree and the index file.
3. The paths in the working tree that are not tracked by GIT.

Among the above three items, the first item is the one that we commit by using git commit command. Item two and three can be committed only after running git add command.

367. What is the main difference between git diff and git status?

In GIT, git diff shows the differences between different commits or between the working directory and index. Whereas, git status command just shows the current status of the working tree.

368. What is the use of git rm command in GIT?

In GIT, git rm command is used for removing a file from the working tree and the index. We use git rm -r to recursively remove all files from a leading directory.

369. What is the command to apply a stash?

Sometimes we want to save our unfinished work. For this purpose, we use git stash command. Once we want to come back and continue working from the last place where we left, we use git stash apply command to bring back the unfinished work.

So the command to apply a stash is:

git stash apply

Or we can use

git stash apply <stashname>

370. Why do we use git log command?

We use git log command to search for specific commits in project history. We can search git history by author, date, or content. It can even list the commits that were done x days before or after a specific date.

371. Why do we need git add command in GIT?

GIT gives us a very good feature of staging our changes before commit. To stage the changes, we use git add command. This adds our changes from the working directory to the index. When we are working on multiple tasks and we want to just commit the finished tasks, we first add finished changes to the staging area and then commit it. At this time, git add command is very helpful.

372. Why do we use git reset command?

We use git reset command to reset the current HEAD to a specific state. By default, it reverses the action of git add command. So we use git reset command to undo the changes of git add command.

373. What does a commit object contain?

Whenever we do a commit in GIT by using git commit command, GIT creates a new commit object. This commit object is saved to the GIT repository. The commit object contains the following information:

- HASH: The SHA1 hash of the Git tree that refers to the state of the index at commit time.

- Commit Author: The name of the person/process doing the commit and date/time.
 - Comment: Some text messages that contain the reason for the commit.
-

374. How can we convert git log messages to a different format?

We can use the pretty option in git log command for this.

`git log --pretty`

This option converts the output format from default to other formats. There are pre-built formats available for our use.

`git log --pretty=oneline`

E.g.

`git log --pretty=format:"%h - %an, %ar : %s"`

ba72a6c - Dave Adams, 3 years ago : changed the version number

375. What are the programming languages in which git hooks can be written?

Git hooks are generally written in shell and PERL scripts. But these can be written in any other language as long as it has an executable. Git hooks can also be written in Python script.

376. What is a commit message in GIT?

A commit message is a comment that we add to a commit. We can provide meaningful information about the reason for commit by using a commit message. In most organizations, it is mandatory to put a commit message along with each commit. Often, commit messages contain JIRA ticket, bug id, defect id, etc. for a project.

377. How GIT protects the code in a repository?

GIT is made very secure since it contains the source code of an organization. All the objects in a GIT repository are encrypted with a hashing algorithm called SHA1. This algorithm is quite strong and fast. It protects source code and other contents of the repository against possible malicious attacks. This algorithm also maintains the integrity of the GIT repository by protecting the change history against accidental changes.

378. How GIT provides flexibility in version control?

GIT is a very flexible version control system. It supports non-linear development workflows. It supports flows that are compatible with external protocols and existing systems. GIT also supports both branching and tagging that promotes multiple kinds of workflows in version control.

379. How can we change a commit message in GIT?

If a commit has not been pushed to GitHub, we can use `git commit --amend` command to change the commit message. When we push the commit, a new message appears on GitHub.

380. Why is it advisable to create an additional commit instead of amending an existing commit?

Git amend internally creates a new commit and replaces the old commit. If commits have already been pushed to the central repository, it should not be used to modify the previous commits. It should generally be used for only amending the git comment.

381. What is a bare repository in GIT?

A repository created with `git init --bare` command is a bare repository in GIT. The bare repository does not contain any

working or checked out copy of source files. A bare repository stores git revision history in the root folder of the repository instead of in a .git subfolder. It is mainly used for sharing and collaborating with other developers. We can create a bare repository in which all developers can push their code. There is no working tree in a bare repository, since no one directly edits files in a bare repository.

382. How do we put a local repository on GitHub server?

To put a local repository on GitHub, we first add all the files of the working directory into the local repository and commit the changes. After that, we call git remote add command to add the local repository on GitHub server. Once it is added, we use git push command to push the contents of the local repository to the remote GitHub server.

383. How will you delete a branch in GIT?

We use git branch -d command to delete a branch in GIT. In case a local branch is not fully merged, but we want to delete it by force, then we use git branch -D command.

384. How can we set up a Git repository to run code sanity checks and UAT tests just before a commit?

We can use git hooks for this kind of purpose. We can write the code sanity checks in a script. This script can be called by the pre-commit hook of the repository. If this hook passes, then only the commit will be successful.

385. How can we revert a commit that was pushed earlier and is public now?

We can use git revert command for this purpose. Internally, git revert command creates a new commit with patches that reverse the changes done in previous commits. The other option is to checkout a previous commit version and then commit it as a new commit.

386. In GIT, how will you compress last n commits into a single commit?

To compress last n commits into a single commit, we use git rebase command. This command compresses multiple commits and creates a new commit. It overwrites the history of commits. It should be done carefully, since it can lead to unexpected results.

387. How will you switch from one branch to a new branch in GIT?

In GIT, we can use git checkout command to switch to a new branch.

388. How can we clean unwanted files from our working directory in GIT?

GIT provides git clean command to recursively clean the working tree. It removes the files that are not under version control in GIT. If we use git clean -x, then ignored files are also removed.

389. What is the purpose of git tag command?

We use git tag command to add, delete, list, or verify a tag object in GIT. Tag objects created with options -a, -s, -u are also known as annotated tags. Annotated tags are generally used for release.

390. What is cherry-pick in GIT?

A git cherry-pick is a very useful feature in GIT. By using this command, we can selectively apply the changes done by existing commits. In case we want to selectively release a feature, we can remove the unwanted files and apply only selected commits.

391. What is shortlog in GIT?

A shortlog in GIT is a command that summarizes the git log output. The output of git shortlog is in a format suitable for release announcements.

392. How can you find the names of files that were changed in a specific commit?

Every commit in GIT has a hash code. This hash code uniquely represents the GIT commit object. We can use git diff-tree command to list the name of files that were changed in a commit. The command will be as follows:

```
git diff-tree -r <hash of commit>
```

By using -r flag, we just get the list of individual files.

393. How can we attach an automated script to run on the event of a new commit by push command?

In GIT, we can use a hook to run an automated script on a specific event. We can choose between pre-receive, update, or post-receive hook and attach our script on any of these hooks. GIT will automatically run the script on the event of any of these hooks.

394. What is the difference between pre-receive, update and post-receive hooks in GIT?

Pre-receive hook is invoked when a commit is pushed to a destination repository. Any script attached to this hook is executed before updating any reference. This is mainly used to enforce development best practices and policies. Update hook is similar to pre-receive hook. It is triggered just before any updates are done. This hook is invoked once for every commit that is pushed to a destination repository. Post-receive hook is invoked after the updates have been done and accepted by a destination repository. This is mainly used to configure deployment scripts. It can also invoke Continuous Integration (CI) systems and send notification emails to relevant parties of a repository.

395. Do we have to store Scripts for GIT hooks within the same repository?

A Hook is local to a GIT repository. But the script attached to a hook can be created either inside the hooks directory or it can be stored in a separate repository. But we have to link the script to a hook in our local repository. In this way, we can maintain versions of a script in a separate repository, but use them in our repository where hooks are stored. Also, when we store scripts in a separate common repository, we can reuse the same scripts for different purposes in multiple repositories.

396. How can we determine the commit that is the source of a bug in GIT?

In GIT, we can use git bisect command to find the commit that has introduced a bug in the system. GIT bisect command internally uses binary search algorithm to find the commit that introduced a bug. We first tell a bad commit that contains the bug and a good commit that was present before the bug was introduced. Then git bisect picks a commit between those two endpoints and asks us whether the selected commit is good or bad. It continues to narrow down the range until it discovers the exact commit responsible for introducing the bug.

397. How can we see differences between two commits in GIT?

We can use git diff command to see the differences between two commits. The syntax for a simple git diff command to compare two commits is:

```
git diff <commit#1> <commit#2>
```

398. What are the different ways to identify a commit in GIT?

Each commit object in GIT has a unique hash. This hash is a 40 characters checksum hash. It is based on SHA1 hashing algorithm. We can use a hash to uniquely identify a GIT commit. Git also provides support for creating an alias for a commit. This alias is known as refs. Every tag in GIT is a ref. These refs can also be used to identify a commit. Some of the special tags in GIT are HEAD, FETCH_HEAD, and MERGE_HEAD.

399. When we run git branch , how does GIT know the SHA-1 of the last commit?

GIT uses the reference named HEAD for this purpose. The HEAD file in GIT is a symbolic reference to the current branch we are working on. A symbolic reference is not a normal reference that contains a SHA-1 value. A symbolic reference contains a pointer to another reference. When we open head file we see:

```
$ cat .git/HEAD
```

```
ref: refs/heads/master
```

If we run git checkout branchA, Git updates the file to look like this:

```
$ cat .git/HEAD
```

```
ref: refs/heads/branchA
```

400. What are the different types of Tags you can create in GIT?

In GIT, we can create two types of Tags.

- **Lightweight Tag:** A lightweight tag is a reference that never moves. We can make a lightweight tag by running a command similar to following:
 - \$ git update-ref refs/tags/v1.0 dad0dab538c970e37ea1e769cbbde608743bc96d
 - **Annotated Tag:** An annotated tag is more complex object in GIT. When we create an annotated tag, GIT creates a tag object and writes a reference to point to it rather than directly to the commit. We can create an annotated tag as follows:
 - \$ git tag -a v1.1 1d410eabc13591cb07496601ebc7c059dd55bfe9 -m 'test tag'
-

401. How can we rename a remote repository?

We can use command git remote rename for changing the name of a remote repository. This changes the short name associated with a remote repository in your local. Command would look as follows:

```
git remote rename repoOldName repoNewName
```

402. Some people use git checkout and some use git co for checkout. How is that possible?

We can create aliases in GIT for commands by modifying the git configuration. In case of calling git co instead of git checkout we can run following command:

```
git config --global alias.co checkout
```

So the people using git co have made the alias for git checkout in their own environment.

403. How can we see the last commit on each of our branch in GIT?

When we run git branch command, it lists all the branches in our local repository. To see the latest commit associated with each branch, we use option -v. Exact command for this is as follows:

```
git branch -v
```

It lists branches as:

```
issue75 83b576c fix issue
```

* master 7b96605 Merge branch 'issue75'
testing 972ac34 add dave to the developer list

404. Is origin a special branch in GIT?

No, origin is not a special branch in GIT. Branch origin is similar to branch master. It does not have any special meaning in GIT. Master is the default name for a starting branch when we run git init command. Origin is the default name for a remote when we run git clone command. If we run git clone -o myOrigin instead, then we will have myOrigin/master as our default remote branch.

405. How can we configure GIT to not ask for password every time?

When we use HTTPS URL to push, the GIT server asks for username and password for authentication. It prompts us on the terminal for this information. If we don't want to type username/password with every single time push, we can set up a "credential cache". It is kept in memory for a few minutes. We can set it by running:

```
git config --global credential.helper cache
```

406. What are the four major protocols used by GIT for data transfer?

GIT uses following major protocols for data transfer:

- Local
 - HTTP
 - Secure Shell (SSH)
 - Git
-

407. What is GIT protocol?

Git protocol is a mechanism for transferring data in GIT. It is a special daemon. It comes pre-packaged with GIT. It listens on a dedicated port 9418. It provides services similar to SSH protocol. But Git protocol does not support any authentication. So on plus side, this is a very fast network transfer protocol. But it lacks authentication.

408. How can we work on a project where we do not have push access?

In case of projects where we do not have push access, we can just fork the repository. By running git fork command, GIT will create a personal copy of the repository in our namespace. Once our work is done, we can create a pull request to merge our changes on the real project.

409. What is git grep?

GIT is shipped along with a grep command that allows us to search for a string or regular expression in any committed tree or the working directory. By default, it works on the files in your current working directory.

410. How can your reorder commits in GIT?

We can use git rebase command to reorder commits in GIT. It can work interactively and you can also select the ordering of commits.

411. How will you split a commit into multiple commits?

To split a commit, we have to use git rebase command in interactive mode. Once we reach the commit that needs to be split, we reset that commit and take the changes that have been reset. Now we can create multiple commits out of that.

412. What is filter-branch in GIT?

In GIT, filter-branch is another option to rewrite history. It can scrub the entire history. When we have large number of commits, we can use this tool. It gives many options like removing the commit related changes to a specific file from history. You can even set your name and email in the commit history by using filter-branch.

413. What are the three main trees maintained by GIT?

GIT maintains following three trees:

- HEAD: This is the last commit snapshot.
 - Index: This is the proposed next commit snapshot.
 - Working Directory: This is the sandbox for doing changes.
-

414. What are the three main steps of working GIT?

GIT has following three main steps in a simple workflow:

1. Checkout the project from HEAD to Working Directory.
 2. Stage the files from Working Directory to Index.
 3. Commit the changes from Index to HEAD.
-

415. What are ours and theirs merge options in GIT?

In GIT, we get two simple options for resolving merge conflicts: ours and theirs. These options tell the GIT which side to favor in merge conflicts. In ours, we run a command like git merge -Xours branchA. As the name suggests, in ours, the changes in our branch are favored over the other branch during a merge conflict.

416. How can we ignore merge conflicts due to Whitespace?

GIT provides an option ignore-space-change in git merge command to ignore the conflicts related to whitespaces. The command to do so is as follows:

```
git merge -Xignore-space-change whitespace
```

417. What is git blame?

In GIT, git blame is a very good option to find the person who changed a specific line. When we call git blame on a file, it displays the commit and name of a person responsible for making change in that line. Following is a sample:

```
1 $ git blame -L 12,19 HelloWorld.java
```

```
1 ^1822fe2 (Dave Adams 2016-03-15 10:31:28 -0700 12) public class HelloWorld {
2 ^1822fe2 (Dave Adams 2016-03-15 10:31:28 -0700 13)
3 ^1822fe2 (Dave Adams 2016-03-15 10:31:28 -0700 14)     public static void main(String[] args) {
4 af6560e4 (Dave Adams 2016-03-17 21:52:20 -0700 16)         // Prints "Hello, World" to the terminal window.
5 a9eaf55d (Dave Adams 2016-04-06 10:15:08 -0700 17)         System.out.println("Hello, World");
6 af6560e4 (Dave Adams 2016-03-17 21:52:20 -0700 18)     }
7 af6560e4 (Dave Adams 2016-03-17 21:52:20 -0700 19) }
```

418. What is a submodule in GIT?

In GIT, we can create submodules inside a repository by using git submodule command. By using submodule command, we can keep a Git repository as a subdirectory of another Git repository. It allows us to keep our commits to submodule separate from the commits to main Git repository.

419. What do you know about AWS Region?

An AWS Region is a completely independent entity in a geographical area. There are two or more Availability Zones in an AWS Region. Within a region, Availability Zones are connected through low-latency links. Since each AWS Region is isolated from another Region, it provides very high fault tolerance and stability. For launching an EC2 instance, we have to select an AMI within the same region.

420. What are the important components of IAM?

The important components of IAM are as follows:

- **IAM User:** An IAM User is a person or service that will interact with AWS. User can sign into AWS Management Console for performing tasks in AWS.
 - **IAM Group:** An IAM Group is a collection of IAM users. We can specify permission to an IAM Group. This helps in managing large number of IAM users. We can simply add or remove an IAM User to an IAM Group to manage the permissions.
 - **IAM Role:** An IAM Role is an identity to which we give permissions. A Role does not have any credentials (password or access keys). We can temporarily give an IAM Role to an IAM User to perform certain tasks in AWS.
 - **IAM Permission:** In IAM we can create two types of Permissions. Identity based and Resource based. We can create a Permission to access or perform an action on an AWS Resource and assign it to a User, Role or Group. We can also create Permissions on resources like S3 bucket, Glacier vault etc and specify who has access to the resource.
 - **IAM Policy:** An IAM Policy is a document in which we list permissions to specify Actions, Resources and Effects. This document is in JSON format. We can attach a Policy to an IAM User or Group.
-

421. What are the important points about AWS IAM?

Some of the important points about AWS IAM are as follows:

- A new User in IAM does not have any permission.
 - AWS IAM assigns an Access Key and a Secret Access Key to a new User.
 - An Access Key cannot be used to login to AWS Console.
 - We use Access Key to access AWS via APIs or Command Line interface.
 - IAM is a universal application. It is common across all the regions in AWS.
 - When we first setup our AWS account, we get a root account that has complete Admin access.
-

422. What are the important features of Amazon S3?

Some of the important features of Amazon S3 are as follows:

- Amazon S3 provides unlimited storage for files.
- File size in Amazon S3 can vary from 0 Bytes to 5 Terabytes.
- We have to store files in Buckets in Amazon S3.
- In Amazon S3, names of buckets have to be unique globally.
- Amazon S3 is Object Based storage.

423. What is the scale of durability in Amazon S3?

Amazon S3 supports durability at the scale of 99.999999999% of time. This is 9 nines after decimal.

424. What are the Consistency levels supported by Amazon S3?

Amazon S3 supports Read after Write consistency when we create a new object by PUT. It means as soon as we Write a new object, we can access it. Amazon S3 supports Eventual Consistency when we overwrite an existing object by PUT. Eventual Consistency means that the effect of overwrite will not be immediate but will happen after some time. For deletion of an object, Amazon S3 supports Eventual Consistency after DELETE.

425. What are the different tiers in Amazon S3 storage?

Different Storage tiers in Amazon S3 are as follows:

- S3 Standard: In this tier, S3 supports durable storage of files that become immediately available. This is used for frequently used files.
 - S3 Standard - Infrequent Access (IA): In this tier, S3 provides durable storage that is immediately available. But in this tier files are infrequently accessed.
 - S3 Reduced Redundancy Storage (RRS): In this tier, S3 provides the option to customers to store data at lower levels of redundancy. In this case data is copied to multiple locations but not on as many locations as standard S3.
-

426. How will you upload a file greater than 100 megabytes in Amazon S3?

Amazon S3 supports storing objects or files up to 5 terabytes. To upload a file greater than 100 megabytes, we have to use Multipart upload utility from AWS. By using Multipart upload we can upload a large file in multiple parts. Each part will be independently uploaded. It doesn't matter in what order each part is uploaded. It even supports uploading these parts in parallel to decrease overall time. Once all the parts are uploaded, this utility makes these as one single object or file from which the parts were created.

427. What happens to an Object when we delete it from Amazon S3?

Amazon S3 provides DELETE API to delete an object. If the bucket in which the object exists is version controlled, then we can specify the version of the object that we want to delete. The other versions of the Object still exist within the bucket. If we do not specify the version, and just pass the key name, Amazon S3 will delete the object and return the version id. And the object will not appear on the bucket. In case the bucket is Multi-factor authentication (MFA) enabled, then the DELETE request will fail if we do not specify a MFA token.

428. What is the use of Amazon Glacier?

Amazon Glacier is an extremely low cost cloud based storage service provided by Amazon. We mainly use Amazon Glacier for long-term backup purpose. Amazon Glacier can be used for storing data archives for months, years or even decades. It can also be used for long term immutable storage based on regulatory and archiving requirements. It provides Vault Lock support for this purpose. In this option, we write once but can read many times same data. One use case is for storing certificates that can be issued only once and only the original person keeps the main copy.

429. Can we disable versioning on a version-enabled bucket in Amazon S3?

No, we cannot disable versioning on a version-enabled bucket in Amazon S3. We can just suspend the versioning on a bucket in S3. Once we suspend versioning, Amazon S3 will stop creating new versions of the object. It just stores the object with null version ID. On overwriting an existing object, it just replaces the object with null version ID. So any existing versions of the object still remain in the bucket. But there will be no more new versions of the same object except for the null version ID object.

430. What are the use cases of Cross Region Replication Amazon S3?

We can use Cross Region Replication Amazon S3 to make copies of an object across buckets in different AWS Regions. This copying takes place automatically and in an asynchronous mode. We have to add replication configuration on our source bucket in S3 to make use of Cross Region Replication. It will create exact replicas of the objects from source bucket to destination buckets in different regions. Some of the main use cases of Cross Region Replication are as follows:

- **Compliance:** Sometimes there are laws/regulatory requirements that ask for storing data at farther geographic locations. This kind of compliance can be achieved by using AWS Regions that are spread across the world.
 - **Failover:** At times, we want to minimize the probability of system failure due to complete blackout in a region. We can use Cross-Region Replication in such a scenario.
 - **Latency:** In case we are serving multiple geographies, it makes sense to replicate objects in the geographical Regions that are closer to end customer. This helps in reducing the latency.
-

431. Can we do Cross Region replication in Amazon S3 without enabling versioning on a bucket?

No, we have to enable versioning on a bucket to perform Cross Region Replication.

432. What are the different types of actions in Object Lifecycle Management in Amazon S3?

There are mainly two types of Object Lifecycle Management actions in Amazon S3.

- **Transition Actions:** These actions define the state when an Object transitions from one storage class to another storage class. E.g. a new object may transition to STANDARD_IA (infrequent access) class after 60 days of creation. And it can transition to GLACIER after 180 days of creation.
 - **Expiration Actions:** These actions specify what happens when an Object expires. We can ask S3 to delete an object completely on expiration.
-

433. How do we get higher performance in our application by using Amazon CloudFront?

If our application is content rich and used across multiple locations, we can use Amazon CloudFront to increase its performance. Some of the techniques used by Amazon CloudFront are as follows:

Caching: Amazon CloudFront caches the copies of our application's content at locations closer to our viewers. By this caching our users get our content very fast. Also due to caching the load on our main server decreases.

Edge / Regional Locations: CloudFront uses a global network of Edge and Regional edge locations to cache our content. These locations cater to almost all of the geographical areas across the world.

Persistent Connections: In certain cases, CloudFront keeps persistent connections with the main server to fetch the content quickly.

Other Optimization: Amazon CloudFront also uses other optimization techniques like TCP initial congestion window etc to deliver high performance experience.

434. What is the mechanism behind Regional Edge Cache in Amazon CloudFront?

A Regional Edge Cache location lies between the main webserver and the global edge location. When the popularity of an object/content decreases, the global edge location may take it out from the cache. But Regional Edge location maintains a larger cache. Due to this the object/content can stay for long time in Regional Edge location. Due to this CloudFront does not have to go back to main webserver. When it does not find any object in Global Edge location it just looks for in Regional Edge location. This improves the performance for serving content to our users in Amazon CloudFront.

435. What are the benefits of Streaming content?

We can get following benefits by Streaming content:

- **Control:** We can provide more control to our users for what they want to watch. In a video streaming, users can select the locations in video where they want to start watching from.
 - **Content:** With streaming our entire content does not stay at a user's device. Users gets only the part they are watching. Once the session is over, content is removed from the user's device.
 - **Cost:** With streaming there is no need to download all the content to a user's device. A user can start viewing content as soon as some part is available for viewing. This saves costs since we do not have to download a large media file before starting each viewing session.
-

436. What is Lambda@Edge in AWS?

In AWS, we can use Lambda@Edge utility to solve the problem of low network latency for end users. In Lambda@Edge there is no need to provision or manage servers. We can just upload our Node.js code to AWS Lambda and create functions that will be triggered on CloudFront requests. When a request for content is received by CloudFront edge location, the Lambda code is ready to execute. This is a very good option for scaling up the operations in CloudFront without managing servers.

437. What are the different types of events triggered by Amazon CloudFront?

Different types of events triggered by Amazon CloudFront are as follows:

- **Viewer Request:** When an end user or a client program makes an HTTP/HTTPS request to CloudFront, this event is triggered at the Edge Location closer to the end user.
 - **Viewer Response:** When a CloudFront server is ready to respond to a request, this event is triggered.
 - **Origin Request:** When CloudFront server does not have the requested object in its cache, the request is forwarded to Origin server. At this time this event is triggered.
 - **Origin Response:** When CloudFront server at an Edge location receives the response from Origin server, this event is triggered.
-

438. What is Geo Targeting in Amazon CloudFront?

In Amazon CloudFront we can detect the country from where end users are requesting our content. This information can be passed to our Origin server by Amazon CloudFront. It is sent in a new HTTP header. Based on different countries we can generate different content for different versions of the same content. These versions can be cached at different Edge Locations that are closer to the end users of that country. In this way we are able to target our end users based on their geographic locations.

439. What are the main features of Amazon CloudFront?

Some of the main features of Amazon CloudFront are as follows:

- Device Detection
 - Protocol Detection
 - Geo Targeting
 - Cache Behavior
 - Cross Origin Resource Sharing
 - Multiple Origin Servers
 - HTTP Cookies
 - Query String Parameters
 - Custom SSL
-

440. What are the security mechanisms available in Amazon S3?

Amazon S3 is a very secure storage service. Some of the main security mechanisms available in Amazon S3 are as follows:

- Access: When we create a bucket or an object, only the owner gets the access to the bucket and objects.
- Authentication: Amazon S3 also supports user authentication to control who has access to a specific object or bucket.
- Access Control List: We can create Access Control Lists (ACL) to provide selective permissions to users and groups.
- HTTPS: Amazon S3 also supports HTTPS protocol to securely upload and download data from cloud.
- Encryption: We can also use Server Side Encryption (SSE) in Amazon S3 to encrypt data.

441. What are the benefits of Cloud Computing?

There are ten main benefits of Cloud Computing: Flexibility: The businesses that have fluctuating bandwidth demands need the flexibility of Cloud Computing. If you need high bandwidth, you can scale up your cloud capacity. When you do not need high bandwidth, you can just scale down. There is no need to be tied into an inflexible fixed capacity infrastructure. Disaster Recovery: Cloud Computing provides robust backup and recovery solutions that are hosted in cloud. Due to this there is no need to spend extra resources on homegrown disaster recovery. It also saves time in setting up disaster recovery. Automatic Software Updates: Most of the Cloud providers give automatic software updates. This reduces the extra task of installing new software version and always catching up with the latest software installs. Low Capital Expenditure: In Cloud computing the model is Pay as you Go. This means there is very less upfront capital expenditure. There is a variable payment that is based on the usage. Collaboration: In a cloud environment, applications can be shared between teams. This increases collaboration and communication among team members. Remote Work: Cloud solutions provide flexibility of working remotely. There is no on site work. One can just connect from anywhere and start working. Security: Cloud computing solutions are more secure than regular onsite work. Data stored in local servers and computers is prone to security attacks. In Cloud Computing, there are very few loose ends. Cloud providers give a secure working environment to its users. Document Control: Once the documents are stored in a common repository, it increases the visibility and transparency among companies and their clients. Since there is one shared copy, there are fewer chances of discrepancies. Competitive Pricing: In Cloud computing there are multiple players, so they keep competing among themselves and provide very good pricing. This comes out much cheaper compared to other options. Environment Friendly: Cloud computing saves precious environmental resources also. By not blocking the resources and bandwidth.

442. What is On-demand computing in Cloud Computing?

On-demand Computing is the latest model in enterprise systems. It is related to Cloud computing. It means IT resources can be provided on demand by a Cloud provider. In an enterprise system demand for computing resources varies from

time to time. In such a scenario, On-demand computing makes sure that servers and IT resources are provisioned to handle the increase/decrease in demand. A cloud provider maintains a pool of resources. The pool of resources contains networks, servers, storage, applications and services. This pool can serve the varying demand of resources and computing by various enterprise clients. There are many concepts like grid computing, utility computing, autonomic computing etc. that are similar to on-demand computing. This is the most popular trend in computing model as of now.

443. What are the different layers of Cloud computing?

Three main layers of Cloud computing are as follows:

Infrastructure as a Service (IAAS): IAAS providers give low-level abstractions of physical devices. Amazon Web Services (AWS) is an example of IAAS. AWS provides EC2 for computing, S3 buckets for storage etc. Mainly the resources in this layer are hardware like memory, processor speed, network bandwidth etc.

Platform as a Service (PAAS): PAAS providers offer managed services like Rails, Django etc. One good example of PAAS is Google App Engineer. These are the environments in which developers can develop sophisticated software with ease. Developers just focus on developing software, whereas scaling and performance is handled by PAAS provider.

Software as a Service (SAAS): SAAS provider offer an actual working software application to clients. Salesforce and Github are two good examples of SAAS. They hide the underlying details of the software and just provide an interface to work on the system. Behind the scenes the version of Software can be easily changed.

444. What resources are provided by Infrastructure as a Service (IAAS) provider?

An IAAS provider can give physical, virtual or both kinds of resources. These resources are used to build cloud. IAAS provider handles the complexity of maintaining and deploying these services. IAAS provider also handles security and backup recovery for these services. The main resources in IAAS are servers, storage, routers, switches and other related hardware etc.

445. What is the benefit of Platform as a Service?

Platform as a service (PaaS) is a kind of cloud computing service. A PaaS provider offers a platform on which clients can develop, run and manage applications without the need of building the infrastructure. In PaaS clients save time by not creating and managing infrastructure environment associated with the app that they want to develop.

446. What are the main advantages of PaaS?

The advantages of PaaS are:

- It allows development work on higher level programming with very less complexity.
 - Teams can focus on just the development of the application that makes the application very effective.
 - Maintenance and enhancement of the application is much easier.
 - It is suitable for situations in which multiple developers work on a single project but are not co-located.
-

447. What is the main disadvantage of PaaS?

Biggest disadvantage of PaaS is that a developer can only use the tools that PaaS provider makes available. A developer cannot use the full range of conventional tools. Some PaaS providers lock in the clients in their platform. This also decreases the flexibility of clients using PaaS.

448. What are the different deployment models in Cloud computing?

Cloud computing supports following deployment models:

Private Cloud: Some companies build their private cloud. A private cloud is a fully functional platform that is owned, operated and used by only one organization. Primary reason for private cloud is security. Many companies feel secure in private cloud. The other reasons for building private cloud are strategic decisions or control of operations. There is also a concept of Virtual Private Cloud (VPC). In VPC, private cloud is built and operated by a hosting company. But it is exclusively used by one organization.

Public Cloud: There are cloud platforms by some companies that are open for general public as well as big companies for use and deployment. E.g. Google Apps, Amazon Web Services etc. The public cloud providers focus on layers and application like cloud application, infrastructure management etc. In this model resources are shared among different organizations.

Hybrid Cloud: The combination of public and private cloud is known as Hybrid cloud. This approach provides benefits of both the approaches—private and public cloud. So it is very robust platform. A client gets functionalities and features of both the cloud platforms. By using Hybrid cloud an organization can create its own cloud as well as they can pass the control of their cloud to another third party.

449. What is the difference between Scalability and Elasticity?

Scalability is the ability of a system to handle the increased load on its current hardware and software resources. In a highly scalable system it is possible to increase the workload without increasing the resource capacity. Scalability supports any sudden surge in the demand/traffic with current set of resources. Elasticity is the ability of a system to increase the workload by increasing the hardware/software resources dynamically. Highly elastic systems can handle the increased demand and traffic by dynamically commission and decommission resources. Elasticity is an important characteristic of Cloud Computing applications. Elasticity means how well your architecture is adaptable to workload in real time. E.g. If in a system, one server can handle 100 users, 2 servers can handle 200 users and 10 servers can handle 1000 users. But in case for adding every X users, if you need 2X the amount of servers, then it is not a scalable design. Let say, you have just one user login every hour on your site. Your one server can handle this load. But, if suddenly, 1000 users login at once, can your system quickly start new web servers on the fly to handle this load? Your design is elastic if it can handle such sudden increase in traffic so quickly.

450. What is Software as a Service?

Software as Service is a category of cloud computing in which Software is centrally hosted and it is licensed on a subscription basis. It is also known as On-demand software. Generally, clients access the software by using a thin-client like a web browser. Many applications like Google docs, Microsoft office etc. provide SaaS model for their software. The benefit of SaaS is that a client can add more users on the fly based on its current needs. And client does not need to install or maintain any software on its premises to use this software.

451. What are the different types of Datacenters in Cloud computing?

Cloud computing consists of different types of Datacenters linked in a grid structure. The main types of Datacenters in Cloud computing are:

Containerized Datacenter: As the name suggests, containerized datacenter provides high level of customization for an organization. These are traditional kind of datacenters. We can choose the different types of servers, memory, network and other infrastructure resources in this datacenter. Also we have to plan temperature control, network management and power management in this kind of datacenter.

Low-Density Datacenters: In a Low-density datacenter, we get high level of performance. In such a datacenter if we increase the density of servers, the issue with power comes. With high density of servers, the area gets heated. In such a scenario, effective heat and power management is done. To reach high level of performance, we have to optimize the number of servers in the datacenter.

452. Explain the various modes of Software as a Service (SaaS) cloud environment?

Software as a Service (SaaS) is used to offer different kinds of software applications in a Cloud environment. Generally these are offered on subscription basis. Different modes of SaaS are:

Simple multi-tenancy: In this setup, each client gets its own resources. These resources are not shared with other clients. It is more secure option, since there is no sharing of resources. But it is an inefficient option, since for each client more money is needed to scale it with the rising demands. Also it takes time to scale up the application in this mode.

Fine grain multi-tenancy: In this mode, the feature provided to each client is same. The resources are shared among multiple clients. It is an efficient mode of cloud service, in which data is kept private among different clients but computing resources are shared. Also it is easier and quicker to scale up the SaaS implementation for different clients.

453. What are the important things to care about in Security in a cloud environment?

In a cloud-computing environment, security is one of the most important aspects. With growing concern of hacking, every organization wants to make its software system and data secure. Since in a cloud computing environment, Software and hardware is not on the premises of an organization, it becomes more important to implement the best security practices. Organizations have to keep their Data most secure during the transfer between two locations. Also they have to keep data secure when it is stored at a location. Hackers can hack into application or they can get an unauthorized copy of the data. So it becomes important to encrypt the data during transit as well as during rest to protect it from unwanted hackers.

454. Why do we use API in cloud computing environment?

Application Programming Interfaces (API) is used in cloud computing environment for accessing many services. APIs are very easy to use. They provide a quick option to create different set of applications in cloud environment. An API provides a simple interface that can be used in multiple scenarios. There are different types of clients for cloud computing APIs. It is easier to serve different needs of multiple clients with APIs in cloud computing environment.

455. What are the different areas of Security Management in cloud?

Different areas of Security management in cloud are as follows:

Identity Management: This aspect creates different level of users, roles and their credentials to access the services in cloud.

Access Control: In this area, we create multiple levels of permissions and access areas that can be given to a user or role for accessing a service in cloud environment.

Authentication: In this area, we check the credentials of a user and confirm that it is the correct user. Generally this is done by user password and multi-factor authentication like verification by a one-time use code on cell phone.

Authorization: In this aspect, we check for the permissions that are given to a user or role. If a user is authorized to access a service, they are allowed to use it in the cloud environment.

456. What are the main cost factors of cloud based data center?

Costs in a Cloud based data center are different from a traditional data center. Main cost factors of cloud based data center are as follows:

Labor cost: We need skilled staff that can work with the cloud-based datacenter that we have selected for our operation. Since cloud is not a very old technology, it may get difficult to get the right skill people for handling cloud based datacenter.

Power cost: In some cloud operations, power costs are borne by the client. Since it is a variable cost, it can increase with

the increase in scale and usage.

Computing cost: The biggest cost in Cloud environment is the cost that we pay to Cloud provider for giving us computing resources. This cost is much higher compared to the labor or power costs.

457. How can we measure the cloud-based services?

In a cloud-computing environment we pay for the services that we use. So main criteria to measure a cloud based service its usage. For computing resource we measure by usage in terms of time and the power of computing resource. For a storage resource we measure by usage in terms of bytes (giga bytes) and bandwidth used in data transfer. Another important aspect of measuring a cloud service is its availability. A cloud provider has to specify the service level agreement (SLA) for the time for which service will be available in cloud.

458. How a traditional datacenter is different from a cloud environment?

In a traditional datacenter the cost of increasing the scale of computing environment is much higher than a Cloud computing environment. Also in a traditional data center, there are not much benefits of scaling down the operation when demand decreases. Since most of the expenditure is in capital spent of buying servers etc., scaling down just saves power cost, which is very less compared to other fixed costs. Also in a Cloud environment there is no need to hire a large number of operations staff to maintain the datacenter. Cloud provider takes care of maintaining and upgrading the resources in Cloud environment. With a traditional datacenter, people cost is very high since we have to hire a large number of technical operation people for in-house datacenter.

459. How will you optimize availability of your application in a Cloud environment?

In a Cloud environment, it is important to optimize the availability of an application by implementing disaster recovery strategy. For disaster recovery we create a backup application in another location of cloud environment. In case of complete failure at a data center we use the disaster recovery site to run the application. Another aspect of cloud environment is that servers often fail or go down. In such a scenario it is important to implement the application in such a way that we just kill the slow server and restart another server to handle the traffic seamlessly.

460. What are the requirements for implementing IaaS strategy in Cloud?

Main requirements to implement IAAS are as follows:

Operating System (OS): We need an OS to support hypervisor in IaaS. We can use open source OS like Linux for this purpose.

Networking: We have to define and implement networking topology for IaaS implementation. We can use public or private network for this.

Cloud Model: We have to select the right cloud model for implementing IaaS strategy. It can be SaaS, PaaS or CaaS.

461. What is Docker?

Docker is Open Source software. It provides the automation of Linux application deployment in a software container. We can do operating system level virtualization on Linux with Docker. Docker can package software in a complete file system that contains software code, runtime environment, system tools, & libraries that are required to install and run the software on a server.

462. What is the difference between Docker image and Docker container?

Docker container is simply an instance of Docker image. A Docker image is an immutable file, which is a snapshot of container. We create an image with build command. When we use run command, an Image will produce a container. In programming language, an Image is a Class and a Container is an instance of the class.

463. How will you remove an image from Docker?

We can use docker rmi command to delete an image from our local system.

Exact command is:

```
% docker rmi <Image Id>
```

If we want to find IDs of all the Docker images in our local system, we can use docker images command.

```
% docker images
```

If we want to remove a docker container then we use docker rm command.

```
% docker rm <Container Id>
```

464. How is a Docker container different from a hypervisor?

In a Hypervisor environment we first create a Virtual Machine and then install an Operating System on it. After that we deploy the application. The virtual machine may also be installed on different hardware configurations. In a Docker environment, we just deploy the application in Docker. There is no OS layer in this environment. We specify libraries, and rest of the kernel is provided by Docker engine. In a way, Docker container and hypervisor are complementary to each other.

465. Can we write compose file in json file instead of yaml?

Yes. Yaml format is a superset of json format. Therefore any json file is also a valid Yaml file. If we use a json file then we have to specify in docker command that we are using a json file as follows:

```
% docker-compose -f docker-compose.json up
```

466. Can we run multiple apps on one server with Docker?

Yes, theoretically we can run multiples apps on one Docker server. But in practice, it is better to run different components on separate containers. With this we get cleaner environment and it can be used for multiple uses.

467. What are the common use cases of Docker?

Some of the common use cases of Docker are as follows:

- Setting up Development Environment: We can use Docker to set the development environment with the applications on which our code is dependent.
 - Testing Automation Setup: Docker can also help in creating the Testing Automation setup. We can setup different services and apps with Docker to create the automation testing environment.
 - Production Deployment: Docker also helps in implementing the Production deployment for an application. We can use it to create the exact environment and process that will be used for doing the production deployment.
-

468. What are the main features of Docker-compose?

Some of the main features of Docker-compose are as follows:

- Multiple environments on same Host: We can use it to create multiple environments on the same host server.
- Preserve Volume Data on Container Creation: Docker compose also preserves the volume data when we create a container.
- Recreate the changed Containers: We can also use compose to recreate the changed containers.
- Variables in Compose file: Docker compose also supports variables in compose file. In this way we can create variations of our containers.

469. What is the most popular use of Docker?

The most popular use of Docker is in build pipeline. With the use of Docker it is much easier to automate the development to deployment process in build pipeline. We use Docker for the complete build flow from development work, test run and deployment to production environment.

470. What is the role of open source development in the popularity of Docker?

Since Linux was an open source operating system, it opened new opportunities for developers who want to contribute to open source systems. One of the very good outcomes of open source software is Docker. It has very powerful features. Docker has wide acceptance due to its usability as well as its open source approach of integrating with different systems.

471. How will you remove all files in current directory? Including the files that are two levels down in a sub-directory.

In Unix we have rm command to remove files and sub-directories. With rm command we have -r option that stands for recursive. The -r option can delete all files in a directory recursively. It means if our current directory structure is as follows:

My_dir

-> Level_1_dir

-> Level_1_dir -> Level_2_dir

-> Level_1_dir -> Level_2_dir -> a.txt

With rm -r * command we can delete the file a.txt as well as sub-directories Level_1_dir and Level_2_dir.

Command:

```
rm -r *
```

The asterisk (*) is a wild card character that stands for all the files with any name.

472. What is the difference between the -v and -x options in Bash shell scripts?

In a BASH Unix shell we can specify the options -v and -x on top of a script as follows:

```
#!/bin/bash -x -v
```

With -x option BASH shell will echo the commands like for, select, case etc. after substituting the arguments and variables. So it will be an expanded form of the command that shows all the actions of the script. It is very useful for debugging a shell script. With -v option BASH shell will echo every command before substituting the values of arguments and variables. In -v option Unix will print each line as it reads. In -v option, If we run the script, the shell prints the entire file and then executes. If we run the script interactively, it shows each command after pressing enter.

473. What is a Filter in Unix command?

In Unix there are many Filter commands like: cat, awk, grep, head, tail, cut etc. A Filter is a software program that takes an input and produces an output, and it can be used in a stream operation.

E.g.

```
cut -d : -f 2 /etc/passwd | grep abc
```

We can mix and match multiple filters to create a complex command that can solve a problem. Awk and Sed are complex filters that provide fully programmable features. Even Data scientists use Unix filters to get the overview of data stored in the files.

474. What is Kernel in Unix operating system?

Kernel is the central core component of a Unix operating system (OS). A Kernel is the main component that can control

everything within Unix OS. It is the first program that is loaded on startup of Unix OS. Once it is loaded it will manage the rest of the startup process. Kernel manages memory, scheduling as well as communication with peripherals like printers, keyboards etc. But Kernel does not directly interact with a user. For a new task, Kernel will spawn a shell and user will work in a shell. Kernel provides many system calls. A software program interacts with Kernel by using system calls. Kernel has a protected memory area that cannot be overwritten accidentally by any process.

475. What is a Shell in Unix OS?

Shell in Unix is a user interface that is used by a user to access Unix services. Generally a Unix Shell is a command line interface (CLI) in which users enter commands by typing or uploading a file. We use a Shell to run different commands and programs on Unix operating system. A Shell also has a command interpreter that can take our commands and send these to be executed by Unix operating system. Some of the popular Shells on Unix are: Korn shell, BASH, C shell etc.

476. What are the different shells in Unix that you know about?

Unix has many flavors of Shell. Some of these are as follows:

- Bourne shell: We use sh for Bourne shell.
 - Bourne Again shell: We use bash to run this shell.
 - Korn shell: We can use ksh for Korn shell.
 - Z shell: The command to use this is zsh.
 - C shell: We use csh to run C shell.
 - Enhanced C shell: tcsh is the command for enhanced C shell.
-

477. What is the first character of the output in ls -l command?

We use ls -l command to list the files and directories in a directory. With -l option we get long listing format. In this format the first character identifies the entry type. The entry type can be one of the following:

- b Block special file
- c Character special file
- d Directory
- l Symbolic link
- s Socket link
- p FIFO
- - Regular file
 -

In general we see d for directory and - for a regular file.

478. What is the difference between Multi-tasking and Multi-user environment?

In a Multi-tasking environment, same user can submit more than one tasks and operating system will execute them at the same time. In a Multi-user environment, more than one user can interact with the operating system at the same time.

479. What is Command Substitution in Unix?

Command substitution is a mechanism by which Shell passes the output of a command as an argument to another

command. We can even use it to set a variable or use an argument list in a for loop.

E.g.

```
rm `cat files_to_delete`
```

In this example `files_to_delete` is a file containing the list of files to be deleted. `cat` command outputs this file and gives the output to `rm` command. `rm` command deletes the files. In general Command Substitution is represented by back quotes `.

480. What is an Inode in Unix?

An Inode is a Data Structure in Unix that denotes a file or a directory on file system. It contains information about file like: location of file on the disk, access mode, ownership, file type etc. Each Inode has a number that is used in the index table. Unix kernel uses Inode number to access the contents of an Inode. We can use `ls -li` command to get the inode number of a file.

481. What is the difference between absolute path and relative path in Unix file system?

Absolute path is the complete path of a file or directory from the root directory. In general root directory is represented by `/` symbol. If we are in a directory and want to know the absolute path, we can use `pwd` command. Relative path is the path relative the current location in directory.

E.g. In a directory structure `/var/user/kevin/mail` if we are in `kevin` directory then `pwd` command will give absolute path as `/var/user/kevin`. Absolute path of mail folder is `/var/user/kevin/mail`. For mail folder `./mail` is the relative path of mail directory from `kevin` folder.

482. What are the main responsibilities of a Unix Shell?

Some of the main responsibilities of a Unix Shell are as follows:

- **Program Execution:** A shell is responsible for executing the commands and script files in Unix. User can either interactively enter the commands in Command Line Interface called terminal or they can run a script file containing a program.
 - **Environment Setup:** A shell can define the environment for a user. We can set many environment variables in a shell and use the value of these variables in our program.
 - **Interpreter:** A shell acts as an interpreter for our scripts. It has a built in programming language that can be used to implement the logic.
 - **Pipeline:** A shell also can hookup a pipeline of commands. When we run multiple commands separated by `|` pipe character, the shell takes the output of a command and passes it to next one in the pipeline.
 - **I/O Redirection:** Shell is also responsible for taking input from command line interface (CLI) and sending the output back to CLI. We use `>`, `<`, `>>` characters for this purpose.
-

483. What is a Shell variable?

A Unix Shell variable is an internal variable that a shell maintains. It is local to that Shell. It is not made available to the parent shell or child shell. We generally use lower case names for shell variables in C shell. We can set the value of a shell variable by `set` command.

E.g.

```
% set max_threads=10
```

To delete a Shell variable we can use `unset` command. To use a Shell variable in a script we use `$` sign in front of the variable name.

E.g.

484. What is a Microservice?

A Microservice is a small and autonomous piece of code that does one thing very well. It is focused on doing well one specific task in a big system. It is also an autonomous entity that can be designed, developed and deployed independently. Generally, it is implemented as a REST service on HTTP protocol, with technology-agnostic APIs. Ideally, it does not share database with any other service.

485. What are the benefits of Microservices architecture?

Microservices provide many benefits. Some of the key benefits are:

- **Scaling:** Since there are multiple Microservices instead of one monolith, it is easier to scale up the service that is being used more. Eg. Let say, you have a Product Lookup service and Product Buy service. The frequency of Product Lookup is much higher than Product Buy service. In this case, you can just scale up the Product Lookup service to run on powerful hardware with multiple servers. Meanwhile, Product Buy service can remain on less powerful hardware.
 - **Resilience:** In Microservice architecture, if your one service goes down, it may not affect the rest of the system. The other parts can keep functioning, business as usual (BAU). Eg. Let say, you have Product Recommendation service and Product Buy service. If Product Recommendation service goes down, the Product Buy service can still keep running.
 - **Technology Mix:** With so many changes in technology everyday, you can keep using the latest technology for your new Microservices. You can adopt new technologies with less risk compared to Monolithic architecture. This is one of the best benefits of Microservices architecture.
 - **Reuse:** Microservices help you in reusing the lessons learnt from one service to another.
 - **Easy Deployment:** Microservices architecture, if done correctly, helps in making the deployment process smooth. If anything goes wrong, it can be rolled back easily and quickly in Microservices.
-

486. What is the role of architect in Microservices architecture?

Architects, in Microservices architecture, play the role of Town planners. They decide in broad strokes about the layout of the overall software system. They help in deciding the zoning of the components. They make sure components are mutually cohesive but not tightly coupled. They need not worry about what is inside each zone. Since they have to remain up to date with the new developments and problems, they have to code with developers to learn the challenges faced in day-to-day life. They can make recommendations for certain tools and technologies, but the team developing a micro service is ultimately empowered to create and design the service. Remember, a micro service implementation can change with time. They have to provide technical governance so that the teams in their technical development follow principles of Microservice. At times they work as custodians of overall Microservices architecture.

487. What is the advantage of Microservices architecture over Service Oriented Architecture (SOA)?

Service Oriented Architecture (SOA) is an approach to develop software by creating multiple services. It creates small parts of services and promotes reusability of software. But SOA development can be slow due to use of things like communication protocols SOAP, middleware and lack of principles. On the other hand, Microservices are agnostic to most of these things. You can use any technology stack, any hardware/middleware, any protocol etc. as long as you follow the principles of Microservices. Microservices architecture also provides more flexibility, stability and speed of development over SOA architecture.

488. Is it a good idea to provide a Tailored Service Template for Microservices development in an organization?

If your organization is using similar set of technologies, then it is a good idea to provide a Service Template that can be tailored by development teams. It can make development faster. Also it can help in promoting adoption of various good practices that are already built into template. But if your organization uses wide variety of technologies, then it may not be wise to produce and maintain a template for each service. Instead of that, it is better to introduce tools that help in maintaining same set of practices related to Microservices among all such technologies. There are many organizations that provide tailored templates for Microservices. Eg. Dropwizard, Karyon etc. You can use these templates to make faster development of services in your organization. Also remember that template code should not promote shared code. This can lead to tight coupling between Microservices.

489. What are the disadvantages of using Shared libraries approach to decompose a monolith application?

You can create shared libraries to increase reuse and sharing of features among teams. But there are some downsides to it. Since shared libraries are implemented in same language, it constrains you from using multiple types of technologies. It does not help you with scaling the parts of system that need better performance. Deployment of shared libraries is same as deployment of Monolith application, so it comes with same deployment issues. Shared libraries introduce shared code that can increase coupling in software.

490. What are the characteristics of a Good Microservice?

Good Microservices have these characteristics:

- Loose coupling: A Microservice knows little about any other service. It is as much independent as possible. The change made in one Microservice does not require changes in other Microservices.
 - Highly cohesive: Microservices are highly cohesive so that each one of them can provide one set of behavior independently.
 - Bounded Context: A Microservice serves a bounded context in a domain and communicates with rest of the domain by using an interface for that Bounded context.
 - Business Capability: Microservices individually add business capability that is part of big picture in organization.
-

491. What is Bounded Context?

A bounded context is like a specific responsibility that is developed within a boundary. In a domain there can be multiple bounded contexts that are internally implemented. Eg. A hospital system can have bounded contexts like: Emergency Ward handling, Regular vaccination, Out patient treatment etc. Within each bounded context, each sub-system can be independently designed and implemented.

492. What are the points to remember during integration of Microservices?

Some of the important points to remember during integration of Microservices are:

- Technology Agnostic APIs: Developing Microservices in a technology agnostic way helps in integration of multiple Microservices. With time, the technology implementation can change but the interface between Microservices can remain same.
- Breaking Changes: Every change in Microservice should not become a breaking change for client. It is better to minimize the impact of a change on an existing client so that existing clients do not have to keep changing their code to adapt to changes in a Microservice.
- Implementation Hiding: Each Microservice should hide its internal implementation details from another one. This helps in minimizing the coupling between Microservices that are integrated for a common solution.
- Simple to use: A Microservice should be simple to use for a consumer, so that the integration points are simpler. It should allow clients to choose their own technology stack.

493. Is it a good idea for Microservices to share a common database?

Sharing a common database between multiple Microservices increases coupling between them. One service can start accessing data tables of another service. This can defeat the purpose of bounded context. So it is not a good idea to share a common database between Microservices.

494. What is the preferred type of communication between Microservices? Synchronous or Asynchronous?

Synchronous communication is a blocking call in which client blocks itself from doing anything else, till the response comes back. In Asynchronous communication, client can move ahead with its work after making an asynchronous call. Therefore client is not blocked. In synchronous communication, a Microservice can provide instant response about success or failure. In real-time systems, synchronous service is very useful. In Asynchronous communication, a service has to react based on the response received in future. Synchronous systems are also known as request/response based. Asynchronous systems are event-based. Synchronous Microservices are not loosely coupled. Depending on the need and critical nature of business domain, Microservices can choose synchronous or asynchronous form of communication.

495. What is the difference between Orchestration and Choreography in Microservices architecture?

In Orchestration, we rely on a central system to control and call various Microservices to complete a task. In Choreography, each Microservice works like a State Machine and reacts based on the input from other parts. Orchestration is a tightly coupled approach for integrating Microservices. But Choreography introduces loose coupling. Also, Choreography based systems are more flexible and easy to change than Orchestration based systems. Orchestration is often done by synchronous calls. But choreography is done by asynchronous calls. The synchronous calls are much simpler compared to asynchronous communication.

496. What are the issues in using REST over HTTP for Microservices?

In REST over HTTP, it is difficult to generate a client stub. Some Web-Servers also do not support all the HTTP verbs like GET, PUT, POST, DELETE etc. Due to JSON or plain text in response, performance of REST over HTTP is better than SOAP. But it is not as good as plain binary communication. There is an overhead of HTTP in each request for communication. HTTP is not well suited for low-latency communications. There is more work in consumption of payload. There may be overhead of serialization, deserialization in HTTP.

497. Can we create Microservices as State Machines?

Yes, Microservices are independent entities that serve a specific context. For that context, the Microservice can work as a State Machine. In a State Machine, there are lifecycle events that cause change in the state of the system. Eg. In a Library service, there is a book that changes state based on different events like: issue a book, return a book, lose a book, late return of a book, add a new book to catalog etc. These events and book can form a state machine for Library Microservice.

